ChorusOS man pages section 2SEG: Virtual Memory Segment Services

Sun Microsystems, Inc.
901 San Antonio Road
Palo Alto, CA 94303-4900
U.S.A.

Part No: 806-3330
December 10, 1999

Contents

PREFACE

Overview

A man page is provided for both the naive user, and sophisticated user who is familiar with the ChorusOS™ operating system and is in need of on-line information. A man page is intended to answer concisely the question “What does it do?” The man pages in general comprise a reference manual. They are not intended to be a tutorial.

The following is a list of sections in the ChorusOS man pages and the information it references:

- *Section 1CC: User Utilities; Host and Target Utilities*
- *Section 1M: System Management Utilities*
- *Section 2DL: System Calls; Data Link Services*
- *Section 2K: System Calls; Kernel Services*
- *Section 2MON: System Calls; Monitoring Services*
- *Section 2POSIX: System Calls; POSIX System Calls*
- *Section 2RESTART: System Calls; Hot Restart and Persistent Memory*
- *Section 2SEG: System Calls; Virtual Memory Segment Services*
- *Section 3FTPD: Libraries; FTP Daemon*
- *Section 3M: Libraries; Mathematical Libraries*
- *Section 3POSIX: Libraries; POSIX Library Functions*
- *Section 3RPC: Libraries; RPC Services*
- *Section 3STDC: Libraries; Standard C Library Functions*
- *Section 3TELD: Libraries; Telnet Services*
- *Section 4CC: Files*

- *Section 5FEA: ChorusOS Features and APIs*
- *Section 7P: Protocols*
- *Section 7S: Services*
- *Section 9DDI: Device Driver Interfaces*
- *Section 9DKI: Driver to Kernel Interface*
- *Section 9DRV: Driver Implementations*

ChorusOS man pages are grouped in Reference Manuals, with one reference manual per section.

Below is a generic format for man pages. The man pages of each manual section generally follow this order, but include only needed headings. For example, if there are no bugs to report, there is no BUGS section. See the `intro` pages for more information and detail about each section, and `man`(1) for more information about man pages in general.

NAME — This section gives the names of the commands or functions documented, followed by a brief description of what they do.

SYNOPSIS — This section shows the syntax of commands or functions. When a command or file does not exist in the standard path, its full pathname is shown. Options and arguments are alphabetized, with single letter arguments first, and options with arguments next, unless a different argument order is required.

The following special characters are used in this section:

[ ] The option or argument enclosed in these brackets is optional. If the brackets are omitted, the argument must be specified.

. . . Ellipses. Several values may be provided for the previous argument, or the previous argument can be specified multiple times, for example, ‘ "`filename` . . ." .

| Separator. Only one of the arguments separated by this character can be specified at time.

{ } Braces. The options and / or arguments enclosed within braces are

	interdependent, such that everything enclosed must be treated as a unit.
FEATURES	This section provides the list of features which offer an interface. An API may be associated with one or more system features. The interface will be available if one of the associated features has been configured.
DESCRIPTION	This section defines the functionality and behavior of the service. Thus it describes concisely what the command does. It does not discuss OPTIONS or cite EXAMPLES.. Interactive commands, subcommands, requests, macros, functions and such, are described under USAGE.
OPTIONS	This lists the command options with a concise summary of what each option does. The options are listed literally and in the order they appear in the SYNOPSIS section. Possible arguments to options are discussed under the option, and where appropriate, default values are supplied.
OPERANDS	This section lists the command operands and describes how they affect the actions of the command.
OUTPUT	This section describes the output - standard output, standard error, or output files - generated by the command.
RETURN VALUES	If the man page documents functions that return values, this section lists these values and describes the conditions under which they are returned. If a function can return only constant values, such as 0 or –1, these values are listed in tagged paragraphs. Otherwise, a single paragraph describes the return values of each function. Functions declared void do not return values, so they are not discussed in RETURN VALUES.
ERRORS	On failure, most functions place an error code in the global variable `errno` indicating why they failed. This section lists alphabetically all error codes a function can generate and describes the conditions that cause each error. When more than one condition can cause the same error, each condition is described in a separate paragraph under the error code.

USAGE	This section is provided as a guidance on use. This section lists special rules, features and commands that require in-depth explanations. The subsections listed below are used to explain built-in functionality: Commands Modifiers Variables Expressions Input Grammar
EXAMPLES	This section provides examples of usage or of how to use a command or function. Wherever possible a complete example including command line entry and machine response is shown. Whenever an example is given, the prompt is shown as `example%` or if the user must be superuser, `example#`. Examples are followed by explanations, variable substitution rules, or returned values. Most examples illustrate concepts from the SYNOPSIS, DESCRIPTION, OPTIONS and USAGE sections.
ENVIRONMENT VARIABLES	This section lists any environment variables that the command or function affects, followed by a brief description of the effect.
EXIT STATUS	This section lists the values the command returns to the calling program or shell and the conditions that cause these values to be returned. Usually, zero is returned for successful completion and values other than zero for various error conditions.
FILES	This section lists all filenames referred to by the man page, files of interest, and files created or required by commands. Each is followed by a descriptive summary or explanation.
SEE ALSO	This section lists references to other man pages, in-house documentation and outside publications.
DIAGNOSTICS	This section lists diagnostic messages with a brief explanation of the condition causing the error.
WARNINGS	This section lists warnings about special conditions which could seriously affect your working conditions. This is not a list of diagnostics.
NOTES	This section lists additional information that does not belong anywhere else on the page. It takes the form of an aside to the user, covering points of special interest. Critical information is never covered here.

BUGS	This section describes known bugs and wherever possible, suggests workarounds.

CHAPTER

Virtual Memory Segment Services

12

NAME

dcAlloc, dcFree – Allocate a data cache for a segment; Free a previously allocated data cache

SYNOPSIS

```
#include <mem/chMem.h>
int dcAlloc(KnSgId sgid, KnExtPxMapper * pxm, VmFlags flags, KnLcId * lcidp);

void dcFree(KnLcId lcid);
```

FEATURES

PXM_EXT

DESCRIPTION

Caution - This system call is strictly reserved for internal use only. It MUST NOT be used by any application.

The *dcAlloc* function allocates a new data cache object associated with the data segment identified by the *sgId* parameter. Two subsequent calls to *dcAlloc* with the same segment identifier will allocate two different data cache objects. It is thus the responsibility of the invoker to insure a one-to- one mapping between data cache objects and data segments. The *sgId* is an opaque value for the nucleus which is only used as an argument within upcalls from the VM to the external Proxy-Mapper managing the data segment. The *pxm* argument specifies the routines which will be used to perform upcalls required for this data segment (see *dcPxmDeclare(2SEG)*). The *flags* argument specifies the required properties of the data segment. It must be either K_NOSWAPOUT if the data cache must not be swapped, or 0. If successful, *dcAlloc* returns K_OK and sets the *lcidp* output argument to the identifier of the newly allocated data cache object, otherwise an error code is returned and the value of the *lcidp* field is undefined. The *lcidp* returned by *dcAlloc* is an opaque for the External Proxy-Mapper and must only be used as an argument for the appropriate nucleus calls. The *dcFree* nucleus call destroys the data cache object specified by the *lcidp* argument. The External Proxy-Mapper can destroy a data cache object only if it is empty (it does not contain any physical pages) and the corresponding data segment is used neither by the Proxy-Mapper nor by the VM.

RESTRICTIONS

The current implementation is only applicable to trusted supervisor actors.

RETURN VALUE

If successful K_OK is returned, otherwise a negative error code is returned.

ERRORS

[K_ENOMEM] The system is out of resources.

ATTRIBUTES

See attributes(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

SEE ALSO

dcFlush(2SEG) , dcRead(2SEG) , dcWrite(2SEG) , dcFillZero(2SEG) , dcPgNumber(2SEG) , dcCluster(2SEG) , dcSync(2SEG)

, `dcPxmDeclare`(2SEG) , `rgnMapFromDtCache`(2SEG) , `rgnInitFromDtCache`(2SEG) , `rgnFlush`(2SEG)

NAME

dcCluster – Set the input and output cluster sizes of a data cache

SYNOPSIS

```
#include <mem/chMem.h>
int dcCluster(KnLcId lcid, VmSize incluster, VmSize outcluster);
```

FEATURES

PXM_EXT

DESCRIPTION

Caution - This system call is strictly reserved for internal use only. It MUST NOT be used by any application.

The *dcCluster* function sets the cluster size attributes of the data cache specified by the *lcid* argument. The *incluster* argument specifies the preferred size for both the *pullIn* and *getAcc* up-calls. These invocations will therefore try to use page lists starting on a boundary multiple of *incluster* and of a size equal to *incluster (there is no guarantee it will always be the case).*

Similarly, the *outcluster* argument specifies the preferred size to be used by the nucleus for building page lists to be passed to the proxy-mapper during the *pushOutAsyn* up-call.

The default size in both cases is the page size. The nucleus takes the cluster size into account if it is a power of two and greater than the page size. This call does not acquire any VM lock and therefore cannot be blocked.

RESTRICTIONS

The current implementation is only applicable to trusted supervisor actors.

RETURN VALUE

If successful this call returns K_OK, otherwise the appropriate error code is returned.

ERRORS

[K_EROUND]	At least one of the sizes is not a multiple of the page size.

ATTRIBUTES

See `attributes`(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

SEE ALSO

`dcAlloc`(2SEG), `dcFree`(2SEG), `dcFlush`(2SEG), `dcFillZero`(2SEG), `dcRead`(2SEG), `dcWrite`(2SEG), `dcPgNumber`(2SEG), `dcSync`(2SEG), `dcPxmDeclare`(2SEG), `rgnMapFromDtCache`(2SEG), `rgnInitFromDtCache`(2SEG), `rgnFlush`(2SEG)

NAME

dcFillZero – Fill a data segment with zero

SYNOPSIS

```
#include <mem/chMem.h>
int dcFillZero(KnLcId lcid, VmOffset start, VmOffset end, VmOffset zeroStart);
```

FEATURES

PXM_EXT

DESCRIPTION

Caution - This system call is strictly reserved for internal use only. It MUST NOT be used by any application.

The *dcFillZero* function gives write access rights to the nucleus for the range of bytes between *start* and *end*. The *start* and *end+1* arguments must be "fragment"-aligned. A fragment is defined as the unit used by the VM to manage access rigths for a segment. The size of a fragment is usually set to 512 bytes but may be implementation-dependent. It also fills with zeros the range of bytes between the *startZero* and *end* arguments. The *startZero* argument has no alignment constraints, however, it must be within the range of bytes defined by the *start* and the *end* arguments. The corresponding part of the data cache is marked as modified.

RESTRICTIONS

The current implementation is only applicable to trusted supervisor actors.

RETURN VALUE

If successful K_OK is returned, otherwise a negative error code is returned.

ERRORS

[K_EROUND]	At least one of the arguments is not "fragment"-aligned.
[K_EOFFSET]	The *startZero* offset is not within the range defined by *start*, *end*.

ATTRIBUTES

See `attributes`(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

SEE ALSO

`dcAlloc`(2SEG), `dcFlush`(2SEG), `dcFree`(2SEG), `dcRead`(2SEG), `dcWrite`(2SEG), `dcPgNumber`(2SEG), `dcCluster`(2SEG), `dcSync`(2SEG), `dcPxmDeclare`(2SEG), `rgnMapFromDtCache`(2SEG), `rgnInitFromDtCache`(2SEG), `rgnFlush`(2SEG)

NAME

dcSync, dcFlush – Sync a data cache object; Flush a data cache object

SYNOPSIS

```
#include <mem/chMem.h>
int dcSync(KnLcId lcid, int pagenb, VmOffset * offset);

int dcFlush(KnLcId lcid, KnPxmFlushReq * fldesc, void * pout);
```

FEATURES

PXM_EXT

DESCRIPTION

Caution - This system call is strictly reserved for internal use only. It MUST NOT be used by any application.

The *dcSync* function scans the physical pages attached to the data cache object specified by the *lcid* argument. The scanning is performed incrementally by offset, the start page is defined by the *offset* argument. The *pagenb* defines the maximum number of pages to be scanned. Each modified page will be passed to the *pushOutAsyn* up-call. The way the pages are passed to this upcall depends on the out cluster size as defined by *dcCluster*. The function returns either a non-negative value, which means that all pages have been scanned. The return value is the difference between *pagenb* and the number of scanned pages. A value of -1 means that all pages specified have been scanned but that there are still unscanned pages. In such a case, *dcSync* sets the *offset* output argument to the first unscanned page offset.

The *dcFlush* nucleus call flushes physical pages associated to the data cache object defined by the *lcId* argument, according to the structure pointed to by *fldesc*. This structure is composed of the following members:

```
VmFlags   flags ;
VmOffset  desStart ;
VmOffset  desEnd ;
VmOffset  reqStart ;
VmOffset  reqEnd ;
```

The *dcFlush* call enables the flushing of all pages in the range defined by the *reqStart* and *reqEnd* offsets to be requested. It also permits the specification of a desired range of pages to be flushed, by setting the *desStart* and *desEnd* fields to the appropriate values. If no desired range is needed, the *desStart* field should be set to the value of the *reqStart* field. Similarly, the *desEnd* field should be set the value of the *reqEnd* field. The *desStart* argument should be less than or equal to *reqStart* , and the *desEnd* argument should be greater than or equal to *reqEnd*. The system will flush all pages in the required range, and will try to flush pages in the desired range. In particular, it will avoid being blocked for pages which are in the desired range but not in the required range.

The *dcFlush* call may perform the flush in different ways according to the value of the *flags* field. This field specifies the required flush mode and consists of several parts:

K_FLUSH_ACCREC_NONE	Access rights are left unchanged. This is useful for flushing dirty pages without modifiyng access rights previously granted to the data cache object.
K_FLUSH_ACCREC_WRITE	Write access rights are recalled.
K_FLUSH_ACCREC_READ	Read and write access rights are recalled.
K_FLUSH_ACCREC_INVAL	Read and write access rights are recalled and pages in the desired range are destroyed.
K_FLUSH_POUT_NONE	The *pushOutAsyn* up-call is never invoked. This is useful for modifying access rights without performing any push out operations.
K_FLUSH_POUT_DIRTY	The *pushOutAsyn* up-call is performed for dirty pages only.
K_FLUSH_POUT_ALL	The *pushOutAsyn* up-call is invoked for all pages even if they are not dirty.
K_FLUSH_PAGEFAULT	If this flag is set, *dcFlush* sets the K_PAGEFAULT flag in the *flags* argument of the *pushOutAsyn* up-call.
K_FLUSH_PAGELIST	If this flag is set, *dcFlush* will invoke the *pushOutAsyn* up-call with lists of contiguous pages not aligned on the cluster size defined for the data cache object. If possible, the entire desired range will be passed as a single list to the *pushOutAsyn* up-call. Several lists will be built in the case of discontigous lists of pages (in such a case, several up-calls will be invoked).

The *pout* argument is an opaque for the VM and will be passed back as an argument of the *pushOutAsyn* up-call, if any.

If successful, *dcFlush* returns *K_OK*, otherwise an error code is returned.

The *dcFlush* call acquires an exclusive lock on pages being flushed. For pages belonging to the desired range but not to the required range, *dcFlush* does not block in order to acquire this lock;, if the lock cannot be acquired the page will not be flushed. Pages specified in the *getAcc* up-call are not visible to the *dcFlush* system call. Thus, VM insures that the *dcFlush* operation will never be blocked by a *getAcc* up-call. In other words, there is no risk of deadlock (from a VM point of view) when an Extrenal Proxy-Mapper performs a *dcFlush* while a *getAcc* up-call is blocked in this PXM.

RESTRICTIONS

The current implementation is only applicable to trusted supervisor actors.

RETURN VALUE

If successful *dcSync* returns -1 when all pages specified have been scanned, but there are still unscanned pages. Otherwise, it returns the difference between the number of scanned pages and the *pageNb* argument.

The *dcFlush* call returns *K_OK* in case of success, otherwise an error code is returned.

ERRORS

[K_EROUND]	At least one of the sizes is not a multiple of the page size.

ATTRIBUTES

See `attributes`(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

SEE ALSO

`dcAlloc`(2SEG) , `dcFree`(2SEG) , `dcFlush`(2SEG) , `dcFillZero`(2SEG) , `dcRead`(2SEG) , `dcWrite`(2SEG) , `dcPgNumber`(2SEG) , `dcSync`(2SEG) , `dcPxmDeclare`(2SEG) , `rgnMapFromDtCache`(2SEG) , `rgnInitFromDtCache`(2SEG) , `rgnFlush`(2SEG)

NAME

dcAlloc, dcFree – Allocate a data cache for a segment; Free a previously allocated data cache

SYNOPSIS

```
#include <mem/chMem.h>
int dcAlloc(KnSgId sgid, KnExtPxMapper * pxm, VmFlags flags, KnLcId * lcidp);

void dcFree(KnLcId lcid);
```

FEATURES

PXM_EXT

DESCRIPTION

Caution - This system call is strictly reserved for internal use only. It MUST NOT be used by any application.

The *dcAlloc* function allocates a new data cache object associated with the data segment identified by the *sgId* parameter. Two subsequent calls to *dcAlloc* with the same segment identifier will allocate two different data cache objects. It is thus the responsibility of the invoker to insure a one-to- one mapping between data cache objects and data segments. The *sgId* is an opaque value for the nucleus which is only used as an argument within upcalls from the VM to the external Proxy-Mapper managing the data segment. The *pxm* argument specifies the routines which will be used to perform upcalls required for this data segment (see *dcPxmDeclare(2SEG)*). The *flags* argument specifies the required properties of the data segment. It must be either K_NOSWAPOUT if the data cache must not be swapped, or 0. If successful, *dcAlloc* returns K_OK and sets the *lcidp* output argument to the identifier of the newly allocated data cache object, otherwise an error code is returned and the value of the *lcidp* field is undefined. The *lcidp* returned by *dcAlloc* is an opaque for the External Proxy-Mapper and must only be used as an argument for the appropriate nucleus calls. The *dcFree* nucleus call destroys the data cache object specified by the *lcidp* argument. The External Proxy-Mapper can destroy a data cache object only if it is empty (it does not contain any physical pages) and the corresponding data segment is used neither by the Proxy-Mapper nor by the VM.

RESTRICTIONS

The current implementation is only applicable to trusted supervisor actors.

RETURN VALUE

If successful K_OK is returned, otherwise a negative error code is returned.

ERRORS

[K_ENOMEM] The system is out of resources.

ATTRIBUTES

See attributes(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

SEE ALSO

dcFlush(2SEG) , dcRead(2SEG) , dcWrite(2SEG) , dcFillZero(2SEG) , dcPgNumber(2SEG) , dcCluster(2SEG) , dcSync(2SEG)

, dcPxmDeclare(2SEG) , rgnMapFromDtCache(2SEG) , rgnInitFromDtCache(2SEG) , rgnFlush(2SEG)

NAME

dcGetPages – Get a list of pages for read-ahead purpose

SYNOPSIS

#include <mem/chMem.h>
int **dcGetPages**(KnLcId *lcid*, VmOffset *start*, VmOffset *end*, VmFlags *flags*, KnPage **page*);

FEATURES

PXM_EXT

DESCRIPTION

Caution - This system call is strictly reserved for internal use only. It MUST NOT be used by any application.

The *dcGetPages* function enables a proxy-mapper to implement a read-ahead policy. It scans the data cache identified by the *lcid* argument, looking for pages belonging to the range defined by the *start* and *end* arguments. A new physical page is allocated for each page missing in the data cache. These pages are grouped in a contiguous list whose head is returned to the invoker at the location defined by the argument *page*. They may then be mapped or unmapped using the *pageMap* and *pageUnmap* nucleus calls, respectively. When pages have been filled with meaningful data, the proxy-mapper informs the nucleus using the *pageIoDone* nucleus call.

When I/O is completed, access rights are granted to the nucleus according to the values of the *flags* argument. If *K_READABLE* is set, read access is granted to the nucleus. If *K_WRITABLE* is set write access is granted to the nucleus. Pages returned by the nucleus to the proxy-mapper as a result of the *dcGetPages* nucleus call are write-locked until the read-ahead completes. This insures that concurrent access to these pages (mapping, reading, flushing,...) will be blocked until the read-ahead completes.

If the *K_NOWAITFORMEMORY* value is set in the *flags* argument, the nucleus will not allocate free pages in case free memory is low.

The page list returned by the *dcGetPages* nucleus call starts with the first missing page in the cache and will end at the first non missing page. However, this list cannot be greater than the range specified by the *start* and *end* arguments (which must be page aligned). The effective size of the list is also constrained by the input cluster size of the data cache, except if the *K_PAGELIST* value is set within the *flags argument*.

RESTRICTIONS

The current implementation is only applicable to trusted supervisor actors.

RETURN VALUE

If successful K_OK is returned, otherwise a negative error code is returned.

ERRORS

[K_EROUND]	One of the *start* or *end* arguments is not page aligned.

ATTRIBUTES

See `attributes`(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

SEE ALSO

dcAlloc(2SEG), dcFlush(2SEG), dcFree(2SEG), dcRead(2SEG), dcWrite(2SEG), dcPgNumber(2SEG), dcCluster(2SEG), dcSync(2SEG), dcPxmDeclare(2SEG), rgnMapFromDtCache(2SEG), rgnInitFromDtCache(2SEG), rgnFlush(2SEG)

NAME

dcIsDirty – Test and reset data cache dirty bit

SYNOPSIS

#include <mem/chMem.h>
int **dcIsDirty**(KnLcId *lcid*);

FEATURES

PXM_EXT

DESCRIPTION

Caution - This system call is strictly reserved for internal use only. It MUST NOT be used by any application.

The *dcIsDirty* function returns 0 if the data cache defined by the *lcid* argument has not been modified through mapping. It returns a non–null value if the data cache defined by the *lcid* argument has been dirtied through mapping. The data cache dirty bit is set whenever a dirty mapped page is unloaded from mmu tables or is sync'ed. This may happen when a *rgnFree* nucleus call occurs or when a *dcSync* or *dcFlush* call is invoked. After the dirty bit of the data cache has been tested, it is reset. Thus, a subsequent call to *dcIsDirty* will return a null value. Modification of a data cache through a call to *dcWrite* will not set the data cache dirty bit.

This call is provided mainly to enable filesytems to determine whether a file has been modified through mapping.

RESTRICTIONS

The current implementation is only applicable to trusted supervisor actors.

RETURN VALUE

This function returns 0 if the data cache is not dirty, 1 otherwise.

ERRORS

No error messages are returned.

ATTRIBUTES

See `attributes`(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

SEE ALSO

`dcAlloc`(2SEG), `dcFlush`(2SEG), `dcFree`(2SEG), `dcRead`(2SEG), `dcWrite`(2SEG), `dcPgNumber`(2SEG), `dcCluster`(2SEG), `dcSync`(2SEG), `dcPxmDeclare`(2SEG), `rgnMapFromDtCache`(2SEG), `rgnInitFromDtCache`(2SEG), `rgnFlush`(2SEG)

NAME

dcPgNumber – Get the number of pages for a data cache object

SYNOPSIS

```
#include <mem/chMem.h>
int dcPgNumber(KnLcId lcid);
```

FEATURES

PXM_EXT

DESCRIPTION

Caution - This system call is strictly reserved for internal use only. It MUST NOT be used by any application.

The *dcPgNumber* function returns the number of pages attached to the data cache object specified by the *lcid* argument. This argument should have been acquired via a previous call to *dcAlloc*. This call does not acquire a VM lock and therefore cannot be blocked.

RESTRICTIONS

The current implementation is only applicable to trusted supervisor actors.

RETURN VALUE

This call returns the number of pages.

ERRORS

No error messags are returned.

ATTRIBUTES

See attributes(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

SEE ALSO

dcAlloc(2SEG), dcFree(2SEG), dcFlush(2SEG), dcFillZero(2SEG), dcRead(2SEG), dcWrite(2SEG), dcCluster(2SEG), dcSync(2SEG), dcPxmDeclare(2SEG), rgnInitFromDtCache(2SEG), rgnFlush(2SEG)

NAME

dcPxmDeclare – Initialize an external proxy-mapper descriptor

SYNOPSIS

```
#include <mem/chMem.h>
int dcPxmDeclare(KnExtPxMapper *pxm);
```

FEATURES

PXM_EXT

DESCRIPTION

Caution - This system call is strictly reserved for internal use only. It MUST NOT be used by any application.

The *dcPxmDeclare* function completes the initalization of the proxy-mapper descriptor pointed to by the *pxm* argument. This descriptor can be used later within *dcAlloc* nucleus calls. The proxy-mapper descriptor should have previously been initialized by the invoker. This is done by creating the appropriate laps to fill the following fields of the *KnExtPxMapper* data structure:

```
KnLapDesc   openLap ;
KnLapDesc   closeLap ;
KnLapDesc   getAccLap ;
KnLapDesc   pullInLap ;
KnLapDesc   pushOutAsynLap ;
KnLapDesc   swapOutLap ;
KnLapDesc   statLap ;
```

All local access point descriptors must be valid and remain valid as long as there are data caches managed by a proxy-mapper. As the proxy-mapper descriptor is not copied within the nucleus address space, this descriptor must remain valid and accessible as long as there are data caches managed by a proxy-mapper.

Each of the laps correspond to a possible upcall made by the nucleus to perform actions on the data segments managed by that proxy-mapper. The syntax and semantics of these upcalls are described in the appropriate manual pages.

The *swapOutLap* may be initialised to 0, usually by means of the *lapDescZero* nucleus call. This will prevent the nucleus from performing this upcall at swapout time.

RESTRICTIONS

The current implementation is only applicable to trusted supervisor actors.

RETURN VALUE

K_OK is always returned.

ERRORS

No error messages are returned.

ATTRIBUTES

See `attributes`(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

SEE ALSO

`dcAlloc`(2SEG), `svLapCreate`(2K), `lapDescZero`(2K)

NAME

dcRead, dcWrite – Read data from a data cache; Write data to a data cache

SYNOPSIS

```
#include <mem/chMem.h>
int dcRead(KnLcId lcid, VmOffset start, VmOffset end, KnCap * actcap, VmAddr addr,
void * acclock);

int dcWrite(KnLcId lcid, VmOffset start, VmOffset end, KnCap * actcap, VmAddr addr,
void *acclock);
```

FEATURES

PXM_EXT

DESCRIPTION

Caution - This system call is strictly reserved for internal use only. It MUST NOT be used by any application.

The *dcRead* function reads data from the data cache object named by *lcid* into an actor address space. The source data is specified by the *start* and *end* offsets which define the offset of the first byte to be read and the offset of the last byte to be read. These offsets are related to the data segment associated with the data cache object of a previous *dcAlloc*. The data is copied, starting at the location defined by the *addr* input/output argument, within the actor defined by the *actcap* argument. If *actcap* is set to the value K_MYACTOR the destination actor is the current one. The *addr* argument is updated by the *dcRead* nucleus call, and upon return, points to an address beyond the last read byte upon return. The *acclock* argument is an opaque field for the nucleus, which will be passed to the external Proxy-Mapper as an argument of the *getAcc* up-call, if any.

The *dcWrite* function writes data from an actor address space to the data cache object named by *lcid* .The destination data is specified by the *start* and *end* offsets which define the offset of the first byte to be written and the offset of the last byte to be written. These offsets are related to the data segment associated with the data cache object by a previous *dcAlloc*. The data is copied, starting at the location defined by the *addr* input/output argument, within the actor defined by the *actcap* argument. If *actcap* is set to the value K_MYACTOR the source actor is the current one. The *addr* argument is updated by the *dcWrite* nucleus call, and upon return, points to an address beyond the last byte written. The *acclock* argument is an opaque field for the nucleus, which will be passed to the external Proxy-Mapper as an argument of the *getAcc* up-call, if any.

RESTRICTIONS

The current implementation is only applicable to trusted supervisor actors.

RETURN VALUE

If successful K_OK is returned, otherwise a negative error code is returned. In either case, the *addr* argument is set to an address beyond the last byte read/written.

ERRORS

[K_EADDR]	The address is outside any allocated region.
[K_EFAULT]	Some of the arguments provided are outside the caller's or target's address space.

[K_EINVAL]	An inconsistent actor capability was given.
[K_EOFFSET]	Attempt to access a segment oustside its valid offset range.
[K_EPROT]	Attempt to write a read only data cache.
[K_EUNKNOWN]	*actcap* does not specify a reachable actor.

ATTRIBUTES

See `attributes`(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

SEE ALSO

`dcAlloc`(2SEG) , `dcFree`(2SEG) , `dcFlush`(2SEG) , `dcFillZero`(2SEG) , `dcPgNumber`(2SEG) , `dcCluster`(2SEG) , `dcSync`(2SEG) , `dcPxmDeclare`(2SEG) , `rgnMapFromDtCache`(2SEG) , `rgnInitFromDtCache`(2SEG) , `rgnFlush`(2SEG)

NAME

dcSync, dcFlush – Sync a data cache object; Flush a data cache object

SYNOPSIS

```
#include <mem/chMem.h>
int dcSync(KnLcId lcid, int pagenb, VmOffset * offset);

int dcFlush(KnLcId lcid, KnPxmFlushReq * fldesc, void * pout);
```

FEATURES

PXM_EXT

DESCRIPTION

Caution - This system call is strictly reserved for internal use only. It MUST NOT be used by any application.

The *dcSync* function scans the physical pages attached to the data cache object specified by the *lcid* argument. The scanning is performed incrementally by offset, the start page is defined by the *offset* argument. The *pagenb* defines the maximum number of pages to be scanned. Each modified page will be passed to the *pushOutAsyn* up-call. The way the pages are passed to this upcall depends on the out cluster size as defined by *dcCluster*. The function returns either a non-negative value, which means that all pages have been scanned. The return value is the difference between *pagenb* and the number of scanned pages. A value of -1 means that all pages specified have been scanned but that there are still unscanned pages. In such a case, *dcSync* sets the *offset* output argument to the first unscanned page offset.

The *dcFlush* nucleus call flushes physical pages associated to the data cache object defined by the *lcId* argument, according to the structure pointed to by *fldesc*. This structure is composed of the following members:

```
VmFlags   flags ;
VmOffset  desStart ;
VmOffset  desEnd ;
VmOffset  reqStart ;
VmOffset  reqEnd ;
```

The *dcFlush* call enables the flushing of all pages in the range defined by the *reqStart* and *reqEnd* offsets to be requested. It also permits the specification of a desired range of pages to be flushed, by setting the *desStart* and *desEnd* fields to the appropriate values. If no desired range is needed, the *desStart* field should be set to the value of the *reqStart* field. Similarly, the *desEnd* field should be set the value of the *reqEnd* field. The *desStart* argument should be less than or equal to *reqStart* , and the *desEnd* argument should be greater than or equal to *reqEnd*. The system will flush all pages in the required range, and will try to flush pages in the desired range. In particular, it will avoid being blocked for pages which are in the desired range but not in the required range.

The *dcFlush* call may perform the flush in different ways according to the value of the *flags* field. This field specifies the required flush mode and consists of several parts:

K_FLUSH_ACCREC_NONE	Access rights are left unchanged. This is useful for flushing dirty pages without modifiyng access rights previously granted to the data cache object.
K_FLUSH_ACCREC_WRITE	Write access rights are recalled.
K_FLUSH_ACCREC_READ	Read and write access rights are recalled.
K_FLUSH_ACCREC_INVAL	Read and write access rights are recalled and pages in the desired range are destroyed.
K_FLUSH_POUT_NONE	The *pushOutAsyn* up-call is never invoked. This is useful for modifying access rights without performing any push out operations.
K_FLUSH_POUT_DIRTY	The *pushOutAsyn* up-call is performed for dirty pages only.
K_FLUSH_POUT_ALL	The *pushOutAsyn* up-call is invoked for all pages even if they are not dirty.
K_FLUSH_PAGEFAULT	If this flag is set, *dcFlush* sets the K_PAGEFAULT flag in the *flags* argument of the *pushOutAsyn* up-call.
K_FLUSH_PAGELIST	If this flag is set, *dcFlush* will invoke the *pushOutAsyn* up-call with lists of contiguous pages not aligned on the cluster size defined for the data cache object. If possible, the entire desired range will be passed as a single list to the *pushOutAsyn* up-call. Several lists will be built in the case of discontigous lists of pages (in such a case, several up-calls will be invoked).

The *pout* argument is an opaque for the VM and will be passed back as an argument of the *pushOutAsyn* up-call, if any.

If successful, *dcFlush* returns *K_OK*, otherwise an error code is returned.

The *dcFlush* call acquires an exclusive lock on pages being flushed. For pages belonging to the desired range but not to the required range, *dcFlush* does not block in order to acquire this lock;, if the lock cannot be acquired the page will not be flushed. Pages specified in the *getAcc* up-call are not visible to the *dcFlush* system call. Thus, VM insures that the *dcFlush* operation will never be blocked by a *getAcc* up-call. In other words, there is no risk of deadlock (from a VM point of view) when an Extrenal Proxy-Mapper performs a *dcFlush* while a *getAcc* up-call is blocked in this PXM.

RESTRICTIONS

The current implementation is only applicable to trusted supervisor actors.

RETURN VALUE

If successful *dcSync* returns -1 when all pages specified have been scanned, but there are still unscanned pages. Otherwise, it returns the difference between the number of scanned pages and the *pageNb* argument.

The *dcFlush* call returns *K_OK* in case of success, otherwise an error code is returned.

ERRORS

[K_EROUND]	At least one of the sizes is not a multiple of the page size.

ATTRIBUTES

See `attributes`(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

SEE ALSO

`dcAlloc`(2SEG) , `dcFree`(2SEG) , `dcFlush`(2SEG) , `dcFillZero`(2SEG) , `dcRead`(2SEG) , `dcWrite`(2SEG) , `dcPgNumber`(2SEG) , `dcSync`(2SEG) , `dcPxmDeclare`(2SEG) , `rgnMapFromDtCache`(2SEG) , `rgnInitFromDtCache`(2SEG) , `rgnFlush`(2SEG)

NAME

dcTrunc – Truncate a data segment

SYNOPSIS

```
#include <mem/chMem.h>
int dcTrunc(KnLcId lcid, VmOffset start, VmOffset end, VmOffset zeroStart);
```

FEATURES

PXM_EXT

DESCRIPTION

Caution - This system call is strictly reserved for internal use only. It MUST NOT be used by any application.

The *dcTrunc* call grants write access to the nucleus for the range of bytes between *start* and *end*, and invalidates the *[end+1, K_MAXVMOFFSET]* range of the data cache specified by the *lcid* argument. It then fills with zeros the range of bytes between the *zeroStart* and *end* arguments, and marks the range of bytes defined by the *start*, *end* arguments as modified. The *start* and *end* arguments must be fragment-aligned, and the *zeroStart* argument must be included between these two limits.

Logically, *dcTrunc* is a combination of *dcFlush* and *dcFillZero*. The VM guarantees that the *dcTtrunc* operation will never be blocked by a *getAcc* up-call.

RESTRICTIONS

The current implementation is only applicable to trusted supervisor actors.

RETURN VALUE

If successful K_OK is returned, otherwise a negative error code is returned.

ERRORS

[K_EROUND]	At least one of the arguments is not fragment-aligned.
[K_EOFFSET]	The *startZero* offset is not within the range defined by *start*, *end*.

ATTRIBUTES

See `attributes`(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

SEE ALSO

`dcAlloc`(2SEG), `dcFlush`(2SEG), `dcFillZero`(2SEG), `dcFree`(2SEG), `dcRead`(2SEG), `dcWrite`(2SEG), `dcPgNumber`(2SEG), `dcCluster`(2SEG), `dcSync`(2SEG), `dcPxmDeclare`(2SEG), `rgnMapFromDtCache`(2SEG), `rgnInitFromDtCache`(2SEG), `rgnFlush`(2SEG)

NAME

dcRead, dcWrite – Read data from a data cache; Write data to a data cache

SYNOPSIS

```
#include <mem/chMem.h>
int dcRead(KnLcId lcid, VmOffset start, VmOffset end, KnCap * actcap, VmAddr addr,
void * acclock);

int dcWrite(KnLcId lcid, VmOffset start, VmOffset end, KnCap * actcap, VmAddr addr,
void *acclock);
```

FEATURES

PXM_EXT

DESCRIPTION

Caution - This system call is strictly reserved for internal use only. It MUST NOT be used by any application.

The *dcRead* function reads data from the data cache object named by *lcid* into an actor address space. The source data is specified by the *start* and *end* offsets which define the offset of the first byte to be read and the offset of the last byte to be read. These offsets are related to the data segment associated with the data cache object of a previous *dcAlloc*. The data is copied, starting at the location defined by the *addr* input/output argument, within the actor defined by the *actcap* argument. If *actcap* is set to the value K_MYACTOR the destination actor is the current one. The *addr* argument is updated by the *dcRead* nucleus call, and upon return, points to an address beyond the last read byte upon return. The *acclock* argument is an opaque field for the nucleus, which will be passed to the external Proxy-Mapper as an argument of the *getAcc* up-call, if any.

The *dcWrite* function writes data from an actor address space to the data cache object named by *lcid* .The destination data is specified by the *start* and *end* offsets which define the offset of the first byte to be written and the offset of the last byte to be written. These offsets are related to the data segment associated with the data cache object by a previous *dcAlloc*. The data is copied, starting at the location defined by the *addr* input/output argument, within the actor defined by the *actcap* argument. If *actcap* is set to the value K_MYACTOR the source actor is the current one. The *addr* argument is updated by the *dcWrite* nucleus call, and upon return, points to an address beyond the last byte written. The *acclock* argument is an opaque field for the nucleus, which will be passed to the external Proxy-Mapper as an argument of the *getAcc* up-call, if any.

RESTRICTIONS

The current implementation is only applicable to trusted supervisor actors.

RETURN VALUE

If successful K_OK is returned, otherwise a negative error code is returned. In either case, the *addr* argument is set to an address beyond the last byte read/written.

ERRORS

[K_EADDR]	The address is outside any allocated region.
[K_EFAULT]	Some of the arguments provided are outside the caller's or target's address space.

[K_EINVAL]	An inconsistent actor capability was given.
[K_EOFFSET]	Attempt to access a segment oustside its valid offset range.
[K_EPROT]	Attempt to write a read only data cache.
[K_EUNKNOWN]	*actcap* does not specify a reachable actor.

ATTRIBUTES

See `attributes`(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

SEE ALSO

`dcAlloc`(2SEG) , `dcFree`(2SEG) , `dcFlush`(2SEG) , `dcFillZero`(2SEG) , `dcPgNumber`(2SEG) , `dcCluster`(2SEG) , `dcSync`(2SEG) , `dcPxmDeclare`(2SEG) , `rgnMapFromDtCache`(2SEG) , `rgnInitFromDtCache`(2SEG) , `rgnFlush`(2SEG)

NAME

lcOpen, lcClose, lcCap – Find or create a local cache object for a segment; Release a local cache object; Return the cabability of a local cache

SYNOPSIS

```
#include <mem/chMem.h>
int lcOpen(KnCap * sgcap, VmFlags flags, int * cachelip);

int lcCap(int cacheli, KnCap * lccap);

int lcClose(int cacheli);
```

FEATURES

MEM_VM

DESCRIPTION

Caution - This system call is strictly reserved for internal use only. It MUST NOT be used by any application.

The *lcOpen* call allows the caller to force the kernel to use the same local cache object for the segment specified until a subsequent *lcClose* call is received.

The *sgcap* argument is a pointer to the target segment capablity. The *flags* argument must be zero. This argument will be used in future extensions of the interface.

The *lcOpen* call returns into the variable pointed to by the *cachelip* argument, a local id of the local cache object corresponding to the target segment. The caller can then use the local id in a number of subsequent *lcRead, lcWrite* (see *lcRead(2SEG))* and *lcCap* (see below) calls. The local id must be released using the *lcClose* call and should not be used any more.

The *lcCap* call finds the local cache object specified by its local id (*cacheli* argument) and returns the local cache capability in the variable pointed to by the *lccap* argument.

RESTRICTIONS

The current implementation is only applicable to trusted supervisor actors. An attempt to pass an invalid local id may produce unpredictable system behavior.

RETURN VALUE

If successful K_OK is returned, otherwise a negative error code is returned.

ERRORS

[K_EFAULT]	Some of the arguments provided are outside the caller's address space.
[K_ENOMEM]	The system is out of resources.

ATTRIBUTES

See `attributes`(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

SEE ALSO

`vmStat`(2K) , `MpGetAccess`(2SEG) , `MpPullIn`(2SEG) , `rgnMap`(2SEG) , `lcFlush`(2SEG) , `lcFillZero`(2SEG) , `lcTrunc`(2SEG)

NAME

lcOpen, lcClose, lcCap – Find or create a local cache object for a segment; Release a local cache object; Return the cabability of a local cache

SYNOPSIS

```
#include <mem/chMem.h>
int lcOpen(KnCap * sgcap, VmFlags flags, int * cachelip);

int lcCap(int cacheli, KnCap * lccap);

int lcClose(int cacheli);
```

FEATURES

MEM_VM

DESCRIPTION

Caution - This system call is strictly reserved for internal use only. It MUST NOT be used by any application.

The *lcOpen* call allows the caller to force the kernel to use the same local cache object for the segment specified until a subsequent *lcClose* call is received.

The *sgcap* argument is a pointer to the target segment capablity. The *flags* argument must be zero. This argument will be used in future extensions of the interface.

The *lcOpen* call returns into the variable pointed to by the *cachelip* argument, a local id of the local cache object corresponding to the target segment. The caller can then use the local id in a number of subsequent *lcRead, lcWrite* (see *lcRead(2SEG))* and *lcCap* (see below) calls. The local id must be released using the *lcClose* call and should not be used any more.

The *lcCap* call finds the local cache object specified by its local id (*cacheli* argument) and returns the local cache capability in the variable pointed to by the *lccap* argument.

RESTRICTIONS

The current implementation is only applicable to trusted supervisor actors. An attempt to pass an invalid local id may produce unpredictable system behavior.

RETURN VALUE

If successful K_OK is returned, otherwise a negative error code is returned.

ERRORS

[K_EFAULT]	Some of the arguments provided are outside the caller's address space.
[K_ENOMEM]	The system is out of resources.

ATTRIBUTES

See `attributes`(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

SEE ALSO

`vmStat`(2K) , `MpGetAccess`(2SEG) , `MpPullIn`(2SEG) , `rgnMap`(2SEG) , `lcFlush`(2SEG) , `lcFillZero`(2SEG) , `lcTrunc`(2SEG)

NAME

lcFillZero – zero a range of a local cache

SYNOPSIS

```
#include <mem/chMem.h>
int lcFillZero(KnObjDesc *lcdesc, VmFlags flags, unsigned long ordernb);
```

FEATURES

MEM_VM

DESCRIPTION

Caution - This system call is strictly reserved for internal use only. It MUST NOT be used by any application.

The *lcFillZero* call grants write access rights and zero-fills and marks as modified the [lcdesc->startOffset, lcdesc->startOffset + lcdesc->size) range of the local cache specified by the *lcdesc->dataObject* capability. The *KnObjDesc* structure is described in *lcFlush(2SEG).*

The *lcdesc->startOffset* and *lcdesc->size* fields must be fragment-aligned. The size of the fragment is implementation–dependent and is usually equal to the virtual page size divided by 8 (number of bits in one byte).

The *flags* argument must be zero and will be used in future interface extensions. The *ordernb* argument is ignored.

The *lcFillZero* operation will never be blocked either by a pure *MpGetAccess* or by an impure *MpPullIn* up-call.

RETURN VALUE

If successful K_OK is returned, otherwise a negative error code is returned.

ERRORS

[K_EFAULT]	Some of the arguments provided are outside the caller's address space.
[K_EINVAL]	An inconsistent local cache capability was provided.
[K_EUNKNOWN]	*lcdesc->dataObject* does not specify a reachable local cache.
[K_EROUND]	*lcdesc->startOffset* or *lcdesc->size* is not fragment-aligned.
[K_EOFFSET]	Tried to fill a segment outside the valid offset range in a segment, as returned by *vmStat.*
[K_EINVAL]	The *flags* argument contains invalid flag values.
[K_EFAIL]	An *ipcCall* transaction failed during the remote *lcFillZero.*
[K_EMAPPER]	The mapper doesn't respect the vm/mapper protocol.

ATTRIBUTES

See `attributes`(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

SEE ALSO `MpGetAccess`(2SEG), `MpPullIn`(2SEG), `rgnMap`(2SEG), `lcOpen`(2SEG), `lcFlush`(2SEG), `lcTrunc`(2SEG)

NAME

lcFlush, lcSetRights, sgFlush, sgSyncAll, vmFlush – flush local cache(s)

SYNOPSIS

#include <mem/chMem.h>
int **lcFlush**(KnObjDesc * *lcdesc*, VmFlags *flags*, unsigned long *ordernb*);

int **lcSetRights**(KnObjDesc * *lcdesc*, VmFlags *flags*, unsigned long *ordernb*);

int **sgFlush**(KnObjDesc * *segdesc*, VmFlags *flags*);

int **sgSyncAll**(void);

int **vmFlush**(KnCap * *actorcap*, VmAddr *address*, VmSize *size*, VmFlags *flags*);

FEATURES

MEM_VM

DESCRIPTION

Caution - This system call is strictly reserved for internal use only. It MUST NOT be used by any application.

The *lcFlush* call performs a flush operation on a range of a local cache. The *lcdesc* argument points to a *KnObjDesc* structure whose members are the following:

```
KnCap     dataObject ;
VmOffset  startOffset ;
VmSize    size ;
```

where the *dataObject* field is the local cache capability, the *startOffset* field is the range start offset in the local cache, and the `size` field is the range size. Both *startOffset* and `size` must be fragment-aligned. The size of the fragment is implementation–dependent and is usually equal to the virtual page size divided by 8 (number of bits in one byte).

For *lcFlush*, the target local cache is directly specified by its capability; the flush operation is applied even if the cache is managed by a remote site.

The *sgFlush* call takes the same arguments, except the *dataObject* field specifies the capability of the cached segment. For *sgFlush*, the target local cache is indirectly specified by the capability of the corresponding segment; the flush operation implicitly applies to the segment's cache(s) located on the caller's site.

The *vmFlush* call performs a flush operation on the ranges of the local caches mapped to an address range of an actor address space. The address range must be page-aligned.

The target actor is specified by *actorcap* - a pointer to the actor capability. If *actorcap* is K_MYACTOR, the address space of the current actor is used. If *actorcap* is K_SVACTOR, the supervisor address space is used.

The *flags* argument specifies the mode of the flush and the upper access rights for the target parts after the flush operation.

If the K_COPYBACK flag is set, *xxFlush* writes back any (even clean) cached data of the target parts and keeps the parts access rights unchanged.

If the K_WRITABLE flag is set, *xxFlush* writes back all modified data of the target parts and keeps the parts access rights unchanged.

If the K_READABLE flag is set, *xxFlush* writes back all modified data of the target parts and then sets the parts to read access only.

If the K_FREEZE flag is set, *xxFlush* writes back all modified data of the target parts and then sets the parts as non-accessible.

If the K_NOACCESS flag is set, *xxFlush* writes back all modified data of the target parts and invalidates them.

If the K_DESTROY flag is set, *xxFlush* invalidates the target parts without writing back.

If the K_ASYNC flag is set, the vm performs the write operations required by the *xxFlush* asynchronously.

If K_PAGEFAULT flag is set, *xxFlush* is a result of a page fault. This type of *lcFlush* operation could break the atomicity of a sgRead / sgWrite operation running concurrently on the same local cache parts.

The K_ORDERED flag can only be used with an *lcFlush* or an *lcSetRights* request. If the flags is set, the *orderNb* argument specifies the *lcFlush* message order number. It allows the kernel to detect the situation when two mesages sent by a mapper in one order are received by the kernel in another. For instance, if the mapper grants certain access rights in a *MpGetAccess* reply first, then, processing another request, calls an *lcFlush* to recall the access rights, considering the order numbers, the kernel is aware that the access returned by the *MpGetAccess* reply was already recalled by the mapper, and performs another *MpGetAccess* request.

The *lcSetRights* operation is similar to the *lcFlush* operation, except that it only changes data protections without invalidation and / or writing back: the K_WRITABLE flag is ignored, the K_READABLE flag sets the target parts read—only, whereas the K_FREEZE and K_NOACCESS flags set the target parts to non-accessible. The K_COPYBACK, K_ASYNC and K_DESTROY flags are prohibited.

The *sgSyncAll* operation writes back asynchronously all dirty parts of all local caches on the site, except the local caches mapped at least once to a region with the K_NOSYNC attribute (see *rgnMap(2SEG)).*

RETURN VALUE

If successful K_OK is returned, otherwise a negative error code is returned.

ERRORS

[K_EFAULT]	Some of the arguments provided are outside the caller’s address space.

[K_EINVAL]	An inconsistent actor capability was provided.
[K_EUNKNOWN]	*actorcap* does not specify a reachable actor.
[K_EROUND]	*address* or `size` isn't page-aligned.
[K_EROUND]	*lcdesc->startOffset* or *lcdesc->size* isn't fragment-aligned.
[K_EROUND]	*segdesc->startOffset* or *segdesc->size* isn't fragment-aligned.
[K_EADDR]	Some or all the addresses from the target address range are invalid.
[K_EOFFSET]	Tried to flush a segment outside the valid offset range in a segment, as returned by *vmStat*.
[K_EINVAL]	The *flags* argument contains invalid flag values.
[K_EBUSY]	Tried to invalidate or destroy a no–demand (mapped to a region with K_NODEMAND attribute) physical memory.
[K_EFAIL]	An *ipcCall* transaction failed during the remote *lcFlush* or *lcSetRights*.
[K_EMAPPER]	The mapper doesn't respect the vm / mapper protocol.

RESTRICTIONS

The target actor and the current actor must be located on the same site (*vmFlush* only).

The *sgFlush* and *sgSyncAll* calls remain in the interface for backward compatibility. They will be removed in a future release.

ATTRIBUTES

See `attributes`(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

SEE ALSO

`MpGetAccess`(2SEG) , `MpPullIn`(2SEG) , `MpPushOut`(2SEG) , `rgnMap`(2SEG) , `lcOpen`(2SEG) , `lcFillZero`(2SEG) , `lcTrunc`(2SEG)

NAME

lcOpen, lcClose, lcCap – Find or create a local cache object for a segment; Release a local cache object; Return the cabability of a local cache

SYNOPSIS

```
#include <mem/chMem.h>
int lcOpen(KnCap * sgcap, VmFlags flags, int * cachelip);

int lcCap(int cacheli, KnCap * lccap);

int lcClose(int cacheli);
```

FEATURES

MEM_VM

DESCRIPTION

Caution - This system call is strictly reserved for internal use only. It MUST NOT be used by any application.

The *lcOpen* call allows the caller to force the kernel to use the same local cache object for the segment specified until a subsequent *lcClose* call is received.

The *sgcap* argument is a pointer to the target segment capablity. The *flags* argument must be zero. This argument will be used in future extensions of the interface.

The *lcOpen* call returns into the variable pointed to by the *cachelip* argument, a local id of the local cache object corresponding to the target segment. The caller can then use the local id in a number of subsequent *lcRead, lcWrite* (see *lcRead(2SEG))* and *lcCap* (see below) calls. The local id must be released using the *lcClose* call and should not be used any more.

The *lcCap* call finds the local cache object specified by its local id (*cacheli* argument) and returns the local cache capability in the variable pointed to by the *lccap* argument.

RESTRICTIONS

The current implementation is only applicable to trusted supervisor actors. An attempt to pass an invalid local id may produce unpredictable system behavior.

RETURN VALUE

If successful K_OK is returned, otherwise a negative error code is returned.

ERRORS

[K_EFAULT]	Some of the arguments provided are outside the caller's address space.
[K_ENOMEM]	The system is out of resources.

ATTRIBUTES

See `attributes`(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

SEE ALSO

`vmStat`(2K) , `MpGetAccess`(2SEG) , `MpPullIn`(2SEG) , `rgnMap`(2SEG) , `lcFlush`(2SEG) , `lcFillZero`(2SEG) , `lcTrunc`(2SEG)

NAME

lcPushData – push data from a source local cache to a target local cache

SYNOPSIS

#include <mem/chMem.h>
#include <mem/chMapper.h>
int **lcPushData**(KnObjDesc **source*, VmFlags *sourceFlags*, KnObjDesc **target*, VmFlags *targetFlags*, MpPullInId *pullInId*, unsigned long *ordinalNb*);

FEATURES

MEM_VM

DESCRIPTION

Caution - This system call is strictly reserved for internal use only. It MUST NOT be used by any application.

The `source` argument points to a *KnObjDesc* structure whose members are the following:

```
KnCap     dataObject ;
VmOffset  startOffset ;
VmSize    size ;
```

where the *dataObject* field is the local cache capability, the *startOffset* field is the start offset of the part in the local cache, and the `size` field is the part size. The *sourceFlags* parameter indicates the new access level for the local cache which is the source cache for the push data operation.

If the K_READABLE flag is set, *lcPushData* sends the requested data to the target local cache and sets the source access to read–only.

If the K_NOACCESS flag is set, *lcPushData* sends the requested data to the target local cache and discards the source data.

If the K_PAGEFAULT flag is set, *lcPushData* is the result of a page fault. This operation could break the atomicity of an*sgRead/sgWrite* operation running concurrently on the same local cache parts.

If the K_ORDERED flag is set, the *lcPushData* request is an ordered message.

The *target* argument points to a *KnObjDesc* structure which contains the capability of a local cache which is requesting data for an overlapping area of the same segment. The *targetFlags* parameter can be used to pass access information to the target site. The *pullinId* argument is provided in a *MpIn* request from the target site.

The *ordinalNb* argument specifies the order number of the request. For each local cache the kernel maintains a variable containing the order number expected for the next ordered message. The variable is initialized with zero. When the kernel receives an ordered message for a local cache, it compares the message order number with the order number expected. If the message order number is greater than or equal to the one expected, the kernel sets the expected order number to the message order number plus one and then processes the message. If the

message order number is less than the one expected, the *lcPushData* request is processed, but the expected order number is not modified.

This operation is applied even if the source cache is managed by a remote site. The effect of *lcPushData* on the source cache is similar to *lcFlush* using the same flags. The difference is that *lcFlush* causes modified data to be pushed to the mapper.

RETURN VALUE

If successful, *lcPushData* returns K_OK, otherwise a negative error code is returned. The number of bytes pushed to the target cache is returned in *target->size*.

RESTRICTIONS

The current implementation of *lcPushData* restricts the pushed data to one page.

ERRORS

[K_EBUSY]	Tried to flush a part of a segment locked in memory.

ATTRIBUTES

See `attributes`(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

SEE ALSO

`lcFlush`(2SEG), `MpPullIn`(2SEG)

NAME

lcRead, lcWrite – Read data through a local cache into an actor address space; Write data from an actor address space through a local cache

SYNOPSIS

#include <mem/chMem.h>
int **lcRead**(int *srccacheli*, VmOffset *srcoffseti*, KnCap * *dstactorcap*, VmAdd *dstaddress*, VmSize * *sizep*);

int **lcWrite**(int *dstcacheli*, VmOffset *dstoffset*, KnCap * *srcactorcap*, VmAddr *srcaddress*, VmSize * *sizep*);

FEATURES

MEM_VM

DESCRIPTION

Caution - This system call is strictly reserved for internal use only. It MUST NOT be used by any application.

The *lcRead* function reads data from a segment through a local cache into the actor address space specified by *dstactorcap* - a pointer to the actor capability. If *dstactorcap* is K_MYACTOR, the address space of the current actor is used. If *dstactorcap* is K_SVACTOR, the supervisor address space is used.

The source data is specified by the *srccacheli* argument, which is the local id of the local cache corresponding to the source segment and the *srcoffset* argument, which is the data start offset within the segment. The destination is specified by the *dstaddress* argument, which is the start address in the destination actor address space.

The *lcWrite* function writes data from an actor address space into a segment through a local cache. The source address space is specified by *srcactorcap* , which is a pointer to the actor capability. If *srcactorcap* is K_MYACTOR, the address space of the current actor is used. If *srcactorcap* is K_SVACTOR, the supervisor address space is used.

The source data is specified by the *srcaddress* argument, which is the data start address in the source actor address space. The destination is specified by the *dstcacheli* argument, which is the local id of the local cache corresponding to the destination segment and the *dstoffset* argument, which is the data start offset within the segment.

The data size is specified by the *sizep* argument - a pointer to the variable containing the size of data.

If any error occurs during an *lcRead* or *lcWrite* operation, the number of bytes read or written before the error occurrred is returned to the variable pointed to by the *sizep* argument.

RETURN VALUE

If successful K_OK is returned, otherwise a negative error code is returned.

ERRORS

[K_EFAULT]	Some of the arguments provided are outside the caller's address space.

[K_EINVAL]	An inconsistent actor capability was provided.
[K_EUNKNOWN]	*dstactorcap* or *srcactorcap* does not specify a reachable actor.
[K_EOFFSET]	Attempted to access a segment outside its valid offset range as returned by *vmStat*.
[K_EFAULT]	Attempted to write to a read–only region.
[K_EFAULT]	The read or write buffer is outside any allocated region.
[K_EMAPPER]	The mapper doesn't respect the vm / mapper protocol.

RESTRICTIONS

The caller, the target actor and the local cache must be located on the same site.

ATTRIBUTES

See `attributes`(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

SEE ALSO

`lcOpen`(2SEG) , `lcClose`(2K)

NAME

lcFlush, lcSetRights, sgFlush, sgSyncAll, vmFlush – flush local cache(s)

SYNOPSIS

```
#include <mem/chMem.h>
int lcFlush(KnObjDesc * lcdesc, VmFlags flags, unsigned long ordernb);

int lcSetRights(KnObjDesc * lcdesc, VmFlags flags, unsigned long ordernb);

int sgFlush(KnObjDesc * segdesc, VmFlags flags);

int sgSyncAll(void);

int vmFlush(KnCap * actorcap, VmAddr address, VmSize size, VmFlags flags);
```

FEATURES

MEM_VM

DESCRIPTION

Caution - This system call is strictly reserved for internal use only. It MUST NOT be used by any application.

The *lcFlush* call performs a flush operation on a range of a local cache. The *lcdesc* argument points to a *KnObjDesc* structure whose members are the following:

```
KnCap     dataObject ;
VmOffset  startOffset ;
VmSize    size ;
```

where the *dataObject* field is the local cache capability, the *startOffset* field is the range start offset in the local cache, and the `size` field is the range size. Both *startOffset* and `size` must be fragment-aligned. The size of the fragment is implementation–dependent and is usually equal to the virtual page size divided by 8 (number of bits in one byte).

For *lcFlush*, the target local cache is directly specified by its capability; the flush operation is applied even if the cache is managed by a remote site.

The *sgFlush* call takes the same arguments, except the *dataObject* field specifies the capability of the cached segment. For *sgFlush*, the target local cache is indirectly specified by the capability of the corresponding segment; the flush operation implicitly applies to the segment's cache(s) located on the caller's site.

The *vmFlush* call performs a flush operation on the ranges of the local caches mapped to an address range of an actor address space. The address range must be page-aligned.

The target actor is specified by *actorcap* - a pointer to the actor capability. If *actorcap* is K_MYACTOR, the address space of the current actor is used. If *actorcap* is K_SVACTOR, the supervisor address space is used.

The *flags* argument specifies the mode of the flush and the upper access rights for the target parts after the flush operation.

If the K_COPYBACK flag is set, *xxFlush* writes back any (even clean) cached data of the target parts and keeps the parts access rights unchanged.

If the K_WRITABLE flag is set, *xxFlush* writes back all modified data of the target parts and keeps the parts access rights unchanged.

If the K_READABLE flag is set, *xxFlush* writes back all modified data of the target parts and then sets the parts to read access only.

If the K_FREEZE flag is set, *xxFlush* writes back all modified data of the target parts and then sets the parts as non-accessible.

If the K_NOACCESS flag is set, *xxFlush* writes back all modified data of the target parts and invalidates them.

If the K_DESTROY flag is set, *xxFlush* invalidates the target parts without writing back.

If the K_ASYNC flag is set, the vm performs the write operations required by the *xxFlush* asynchronously.

If K_PAGEFAULT flag is set, *xxFlush* is a result of a page fault. This type of *lcFlush* operation could break the atomicity of a sgRead / sgWrite operation running concurrently on the same local cache parts.

The K_ORDERED flag can only be used with an *lcFlush* or an *lcSetRights* request. If the flags is set, the *orderNb* argument specifies the *lcFlush* message order number. It allows the kernel to detect the situation when two mesages sent by a mapper in one order are received by the kernel in another. For instance, if the mapper grants certain access rights in a *MpGetAccess* reply first, then, processing another request, calls an *lcFlush* to recall the access rights, considering the order numbers, the kernel is aware that the access returned by the *MpGetAccess* reply was already recalled by the mapper, and performs another *MpGetAccess* request.

The *lcSetRights* operation is similar to the *lcFlush* operation, except that it only changes data protections without invalidation and / or writing back: the K_WRITABLE flag is ignored, the K_READABLE flag sets the target parts read—only, whereas the K_FREEZE and K_NOACCESS flags set the target parts to non-accessible. The K_COPYBACK, K_ASYNC and K_DESTROY flags are prohibited.

The *sgSyncAll* operation writes back asynchronously all dirty parts of all local caches on the site, except the local caches mapped at least once to a region with the K_NOSYNC attribute (see *rgnMap(2SEG)*).

RETURN VALUE

If successful K_OK is returned, otherwise a negative error code is returned.

ERRORS

[K_EFAULT]	Some of the arguments provided are outside the caller's address space.

[K_EINVAL]	An inconsistent actor capability was provided.
[K_EUNKNOWN]	*actorcap* does not specify a reachable actor.
[K_EROUND]	*address* or `size` isn't page-aligned.
[K_EROUND]	*lcdesc->startOffset* or *lcdesc->size* isn't fragment-aligned.
[K_EROUND]	*segdesc->startOffset* or *segdesc->size* isn't fragment-aligned.
[K_EADDR]	Some or all the addresses from the target address range are invalid.
[K_EOFFSET]	Tried to flush a segment outside the valid offset range in a segment, as returned by *vmStat*.
[K_EINVAL]	The *flags* argument contains invalid flag values.
[K_EBUSY]	Tried to invalidate or destroy a no–demand (mapped to a region with K_NODEMAND attribute) physical memory.
[K_EFAIL]	An *ipcCall* transaction failed during the remote *lcFlush* or *lcSetRights*.
[K_EMAPPER]	The mapper doesn't respect the vm / mapper protocol.

RESTRICTIONS

The target actor and the current actor must be located on the same site (*vmFlush* only).

The *sgFlush* and *sgSyncAll* calls remain in the interface for backward compatibility. They will be removed in a future release.

ATTRIBUTES

See `attributes`(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

SEE ALSO

`MpGetAccess`(2SEG) , `MpPullIn`(2SEG) , `MpPushOut`(2SEG) , `rgnMap`(2SEG) , `lcOpen`(2SEG) , `lcFillZero`(2SEG) , `lcTrunc`(2SEG)

NAME

lcStat, sgStat – get the statistics of a local cache

SYNOPSIS

```
#include <mem/chMem.h>
#include <mem/chMapper.h>
int lcStat(KnCap * lccap, KnLcStat * stat);

int sgStat(KnCap * segcap, KnLcStat * stat);
```

FEATURES

MEM_VM

DESCRIPTION

Caution - This system call is strictly reserved for internal use only. It MUST NOT be used by any application.

The *lcStat* and *sgStat* functions get the statistics of the local cache specified either by *lccap* - a local cache capability, or by *segcap* - a segment capability. In the latter case, the statistics of the cache of the segment on the current site are returned.

The *KnLcStat* structure describes the statistics associated with a local cache, as follows:

```
VmSize   physMem ;
VmSize   lockMem ;
KnCap    segcap ;
KnCap    lccap ;
```

The *physMem* field specifies the physical memory size currently allocated for the local cache.

The *lockMem* field specifies the physical memory size currently fixed for the local cache.

The *segcap* field specifies the capability of the corresponding segment. It is returned by the *lcStat* call only when the caller is a system actor or when the current thread executes in privileged mode.

The *lccap* field specifies the capability of the segment's local cache on the current site. It is returned by the *sgStat* call only when the caller is a system actor or when the current thread executes in privileged mode.

RETURN VALUE

If successful K_OK is returned, otherwise a negative error code is returned.

ERRORS

[K_EFAULT]	Some of the arguments provided are outside the caller's address space.
[K_EUNDEF]	The segment specified is not cached on the site.

ATTRIBUTES

See `attributes`(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

SEE ALSO `lcFlush`(2SEG)

NAME

lcTrunc – shape the end of a local cache

SYNOPSIS

#include <mem/chMem.h>
int **lcTrunc**(KnObjDesc **lcdesc*, VmOffset *zerooffset*, VmFlags *flags*, unsigned long *ordernb*);

FEATURES

MEM_VM

DESCRIPTION

Caution - This system call is strictly reserved for internal use only. It MUST NOT be used by any application.

The *lcTrunc* function grants write access to the [lcdesc->startOffset, lcdesc->startOffset + lcdesc->size)] range and recalls all access rights from the *[lcdesc->startOffset + lcdesc->size, K_MAXVMOFFSET]* range of the local cache specified by the *lcdesc->dataObject* capability.

Then, *lcTrunc* caches the *[lcdesc->startOffset, zerooffset)* range in the local cache and zeros the *[zerooffset, lcdesc->startOffset + lcdesc->size)* range (if not empty, in other words, *zerooffset == lcdesc->startOffset + lcdesc->size).*

Finally, *lcTrunc* marks the *[lcdesc->startOffset, lcdesc->startOffset + lcdesc->size)* range as modified.

If *lcdesc->size* is equal to zero the *lcTrunc* call simply invalidates the *[lcdesc->startOffset, K_MAXVMOFFSET]* range.

Both *lcdesc->startOffset* and *lcdesc->size* must be fragment-aligned. The size of the fragment is implementation–dependent and usually equal to the virtual page size divided by 8 (number of bits in one byte).

If the K_ORDERED flag is set in the *flags* argument, the *ordinalNumber* argument specifies the *lcTrunc* message order number. It allows the kernel to detect the situation when two mesages sent by a mapper in one order are received by the kernel in another. For instance, if the mapper grants certain access rights in a *MpGetAccess* reply first, then, processing another request, calls an *lcTrunc* to recall the access rights, using the order numbers, the kernel is aware that the access returned by the *MpGetAccess* reply was already recalled by the mapper, and performs another *MpGetAccess* request.

The *lcTrunc* operation cannot be blocked by either a pure *MpGetAccess* or by an impure (combined with a *MpGetAccess) MpPullIn* up-call.

RETURN VALUE

If successful K_OK is returned, otherwise a negative error code is returned.

ERRORS

[K_EFAULT]	Some of the arguments provided are outside the caller's address space.
[K_EINVAL]	An inconsistent local cache capability was provided.
[K_EUNKNOWN]	*lcdesc->dataObject* does not specify a reachable local cache.

[K_EROUND]	*lcdesc->startOffset* or *lcdesc->size* isn't fragment-aligned.
[K_EOFFSET]	Tried to truncate a segment outside the valid offset range in a segment, as returned by *vmStat*.
[K_EOFFSET]	*zeroofset* is out of *[lcdesc->startOffset, lcdesc->startOffset + lcdesc->size]* range.
[K_EINVAL]	The *flags* argument contains invalid flag values.
[K_EBUSY]	Tried to invalidate or destroy a no–demand (mapped to a region with the K_NODEMAND attribute) physical memory.
[K_EFAIL]	An *ipcCall* transaction failed during the remote *lcTrunc*.
[K_EMAPPER]	The mapper doesn't respect the vm / mapper protocol.

ATTRIBUTES

See `attributes`(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

SEE ALSO

`MpGetAccess`(2SEG), `MpPullIn`(2SEG), `rgnMap`(2SEG), `lcOpen`(2SEG), `lcFlush`(2SEG), `lcFillZero`(2SEG)

NAME

lcRead, lcWrite – Read data through a local cache into an actor address space; Write data from an actor address space through a local cache

SYNOPSIS

```
#include <mem/chMem.h>
int lcRead(int srccacheli, VmOffset srcoffseti, KnCap * dstactorcap, VmAdd dstaddress,
VmSize * sizep);

int lcWrite(int dstcacheli, VmOffset dstoffset, KnCap * srcactorcap, VmAddr srcaddress,
VmSize * sizep);
```

FEATURES

MEM_VM

DESCRIPTION

Caution - This system call is strictly reserved for internal use only. It MUST NOT be used by any application.

The *lcRead* function reads data from a segment through a local cache into the actor address space specified by *dstactorcap* - a pointer to the actor capability. If *dstactorcap* is K_MYACTOR, the address space of the current actor is used. If *dstactorcap* is K_SVACTOR, the supervisor address space is used.

The source data is specified by the *srccacheli* argument, which is the local id of the local cache corresponding to the source segment and the *srcoffset* argument, which is the data start offset within the segment. The destination is specified by the *dstaddress* argument, which is the start address in the destination actor address space.

The *lcWrite* function writes data from an actor address space into a segment through a local cache. The source address space is specified by *srcactorcap* , which is a pointer to the actor capability. If *srcactorcap* is K_MYACTOR, the address space of the current actor is used. If *srcactorcap* is K_SVACTOR, the supervisor address space is used.

The source data is specified by the *srcaddress* argument, which is the data start address in the source actor address space. The destination is specified by the *dstcacheli* argument, which is the local id of the local cache corresponding to the destination segment and the *dstoffset* argument, which is the data start offset within the segment.

The data size is specified by the *sizep* argument - a pointer to the variable containing the size of data.

If any error occurs during an *lcRead* or *lcWrite* operation, the number of bytes read or written before the error occurrred is returned to the variable pointed to by the *sizep* argument.

RETURN VALUE

If successful K_OK is returned, otherwise a negative error code is returned.

ERRORS

[K_EFAULT]	Some of the arguments provided are outside the caller's address space.

[K_EINVAL]	An inconsistent actor capability was provided.
[K_EUNKNOWN]	*dstactorcap* or *srcactorcap* does not specify a reachable actor.
[K_EOFFSET]	Attempted to access a segment outside its valid offset range as returned by *vmStat*.
[K_EFAULT]	Attempted to write to a read–only region.
[K_EFAULT]	The read or write buffer is outside any allocated region.
[K_EMAPPER]	The mapper doesn't respect the vm / mapper protocol.

RESTRICTIONS

The caller, the target actor and the local cache must be located on the same site.

ATTRIBUTES

See `attributes`(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

SEE ALSO

`lcOpen`(2SEG) , `lcClose`(2K)

NAME

MpCreate – create a temporary segment at the default mapper

SYNOPSIS

```
#include <mem/chMem.h>
#include <mem/chMapper.h>
MpCreate
request annex(KnMpCreate structure):
   int        service;
   VmFlags    options;
response annex (KnMpCreateReply structure):
   int        diag;
   KnCap      segcap;
   VmFlags    options;
```

FEATURES

MEM_VM

DESCRIPTION

Caution - This system call is strictly reserved for internal use only. It MUST NOT be used by any application.

In order to create a temporary segment at the default mapper, the kernel memory management system performs an *MpCreate* message transaction. In other words, it sends, using *ipcCall*(2K), an *MpCreate* request message to a port or group of ports identifying the mapper.

The request message consists of an annex (no body) whose head matches the *KnMpCreate* structure defined above. The *service* field of this structure must be set to KN_MPCREATE.

The mapper replies with a message, also consisting of an annex only, whose head must match the *KnMpCreateReply* structure. The *diag* field is the operation return code. The *diag* must be either 0 (K_OK) or a negative number. If a negative error code is returned, the kernel returns it to the original kernel call (if any).

The *segcap* field contains the segment capability created.

The *options* field of the return message must be zero for backward compatibility with future versions.

ATTRIBUTES

See `attributes`(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

SEE ALSO

`MpGetAccess`(2SEG), `MpPullIn`(2SEG), `MpPushOut`(2SEG), `MpRelease`(2SEG)

NAME

MpGetAccess – get access to data through a local cache

SYNOPSIS

```
#include <mem/chMem.h>
#include <mem/chMapper.h>
MpGetAccess
request annex (KnMpGetAccess structure):
   int            service ;
   KnKey          segkey ;
   KnCap          lccap ;
   VmOffset       accessOffset ;
   VmSize         accessSize ;
   VmFlags        requiredAccess ;
   VmSize         requiredAccessOffset ;
   VmSize         requiredAccessSize ;
response annex (KnMpGetAccessReply structure):
   int           diag ;
   VmFlags        grantedAccess ;
   unsigned long  ordinalNumber ;
   VmOffset       returnAccessOffset ;
   VmSize         returnAccessSize ;
   VmSize         inClusterSize ;
   VmSize          outClusterSize ;
```

FEATURES

MEM_VM

DESCRIPTION

Caution - This system call is strictly reserved for internal use only. It MUST NOT be used by any application.

In order to demand access rights for a range of a segment through a local cache, the kernel memory management system performs an *MpGetAccess* message transaction. In other words it sends, using *ipcCall*(2K), an *MpGetAccess* request message to the mapper managing the segment. The mapper is identified by the *ui* field of the segment capability.

The request message consists of an annex (no body) whose head matches the *KnMpGetAccess* structure defined above. The *service* field of this structure must be set to KN_MPGETACCESS.

The kernel demands access rights to *accessSize* bytes of data using the *accessOffset*, the starting offset in the segment is specified by *segkey* on the invoked mapper,

for the local cache specified by *lccap*. Both *accessOffset* and *accessSize* are always page-aligned.

The kernel also specifies in the message that access rights for *requiredAccessSize* bytes of data with the *requiredAccessOffset* starting offset is mandatory. The kernel guarantees that *[accessOffset, accessOffset + accessSize]* includes *[requiredAccessOffset, requiredAccessOffset + requiredAccessSize]*.

The kernel can send a get access request when it performs one of the following operations:

- *lc(sg)Read* or a *lc(sg)Write* kernel calls. In this case *[requiredAccessOffset, requiredAccessOffset + requiredAccessSize)* specifies the exact (in bytes) boundaries of the operation.

  The kernel can perform a *lc(sg)Read/Write* call without a *MpGetAccess* transaction if it already has access to the required range.

- any other operation which acts on a range of the segment through a virtual address space as a page fault or *vmLock, vmCopy, rgnSetPaging, rgnInit,* for example.

  In this, and only this, case the K_PAGEFAULT flag is set in the *requiredAccess* field.

  The kernel can ask for access rights to the operation range (the operation range of a page fault is the whole faulty page) using a number of consecutive *MpGetAccess* transactions. The *requiredAccessOffset* of each transaction is page-aligned and *requiredSize % vmPageSize()* is equal to 1 (the last byte of the required range is the first byte of a page).

  The kernel can perform the operation without an *MpGetAccess* request for a particular page overlapping the operation range if it has already the requested access to the whole page.

If the K_WRITABLE flag in the *requiredAccess* field is set, write access to the data is requested, otherwise read-only access is requested.

The mapper must reply with a message, consisting of an annex (no body) whose head matches the *KnMpGetAccessReply* structure.

The *diag* field is the transaction return code. The *diag* field must be either 0 (K_OK), or a negative number. When a negative error code is returned, there are two possibilities:

- *returnAccessSize* is equal to zero. In this case the kernel immediately gives up the processing of the original operation and returns from the original kernel operation with a *diag* error code.

- *returnAccessSize* is positive. In this case the kernel reduces the original operation range until *returnAccessOffset + returnAccessSize,* completes the operation and returns from the original kernel operation with a *diag* error code. Also, if it's allowed by the interface of the operation (*lcRead* or *lcWrite* calls), the number of bytes effectively processed will be returned.

If the original kernel operation is a page fault an error provokes exception processing at the end of the page fault handler execution (see *svExcHandler(2K)).*

The K_MPIVERS bitmask defines a bit-field of the *grantedAccess* field which specifies the version of the *MpGetAccess* protocol supported by the mapper. The protocol version has to be equal to K_MPIVER1.

The mapper grants access to *returnAccessSize* bytes of data with the starting offset in the segment equal to *returnAccessOffset.* The mapper has to guarantee that *[returnAccessOffset, returnAccessOffset + returnAccessSize]* is included in *[accessOffset, accessOffset + accessSize]* but includes *[requiredAccessOffset, requiredAccessOffset + requiredAccessSize].* Note that if *diag* is negative (see above) the last condition can be not respected.

The kernel requires *returnAccessOffset* and *returnAccessSize* to be "fragment"-aligned. The size of the fagment is implementation—dependent but usually equal to the virtual page size devided by 8 (number of bits in one byte).

The alignment requirements above mean that the mapper can return access rights to a partial page. The kernel implements the following policy with respect to pages with partial access rights:

- The kernel maps the page to the target virtual address if required by the target operation. The kernel page fault handler can map a page with partial access rights to the faulty virtual address after the corresponding *MpGetAccess* return. Note that in the case of another page fault on a page with partial access rights, the kernel will call *MpGetAccess* again.
- The kernel doesn't guarantee that the page compliment (the ranges of the page without access rights) will be included in a subsequent *mpPushOut* request even if it was modified using mmu. Neither does the kernel guarantee that the page complement will not be included in any subsequent *mpPushOut* request.

If the mapper grants write access, it sets the K_WRITABLE flag in the *grantedAccess* field, otherwise read only access is granted. The mapper could grant write access in response to a read-only access request, but it must grant write access in response to a write access request.

If the K_ORDERED flag is set in the *grantedAccess* field, the *ordinalNumber* field specifies the reply message order number. It allows the kernel to detect the situation when two mesages sent by a mapper in one order are received by the kernel in another. For instance, suppose the mapper grants access rights in an *MpGetAccess* reply first, and then, processing another request, calls an *lcFlush* to recall the access rights. In this case, the order numbers indicate to the kernel that the access returned by the *MpGetAccess* reply was already recalled by the mapper, and performs a *MpGetAccess* request again.

If the K_STALE flags is set in the *grantedAccess* field, any data already cached from the returned range are considered as stale and will be destroyed by the kernel upon receipt of the reply message.

The mapper may also set the K_RELEASE flag (in the *grantedAccess* field). In this case the memory management has to perform an *MpRelease* request prior to destroying the local cache.

If the K_GETATTR flag is set in the *requiredAccess* field of the request message, the current *MpGetAccess* request is combined with a "get segment attributes" request. When the mapper replies to a combined request it may return the segment clustering attributes. If the mapper sets the K_CLUSTER flag (in the *grantedAccess* field) the *inClusterSize* and *outClusterSize* fields define the segment clustering attributes. The *inClusterSize* defines the optimal (from the mapper point of view) data size for any future *MpPullIn* operation performed on the segment . The *outClusterSize* defines the optimal data size for any future *MpPushOut* operation. Both *inClusterSize* and *outClusterSize* have to be a power of two and page-aligned.

If the kernel detects that the mapper is not adhering to the protocol desscribed above, the K_EMAPPER error is returned to the orginal kernel operation. If the original kernel operation is a page fault an error provokes exception processing at the end of the page fault handler execution (see *svExcHandler(2K)).*

ATTRIBUTES

See `attributes`(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

SEE ALSO

`MpPullIn`(2SEG), `MpPushOut`(2SEG), `rgnMap`(2SEG), `lcOpen`(2SEG), `lcFlush`(2SEG), `lcFillZero`(2SEG), `lcTrunc`(2SEG)

NAME

MpPullIn – read data into a local cache

SYNOPSIS

```
#include <mem/chMem.h>
#include <mem/chMapper.h>
MpPullIn
request annex (KnMpInle structure):
   int         service ;
   KnCap       segkey ;
   KnCap       lccap ;
   VmOffset    accessOffset ;
   VmSize      accessSize ;
   VmFlags     requiredAccess ;
   VmOffset    requiredAccessOffset ;
   VmSize      requiredAccessSize ;
   VmOffset    dataOffset ;
   VmSize      dataSize ;
   VmOffset    requiredDataOffset ;
   VmSize      requiredDataSize ;
   MpPullInId  pullInId ;
response annex (KnMpInReply structure) :
   int         diag ;
   VmFlags     grantedAccess ;
   unsigned    ordinalNumber ;
   VmOffset    returnAccessOffset ;
   VmSize      returnAccessSize ;
   VmSize      inClusterSize ;
   VmSize      outClusterSize ;
   VmOffset    returnDataOffset ;
   VmSize      returnDataSize ;
response body:
   unsigned char  data [bodysize] ;
```

FEATURES

MEM_VM

DESCRIPTION

Caution - This system call is strictly reserved for internal use only. It MUST NOT be used by any application.

In order to demand the contents of a range of a local cache to be updated from the corresponding segment, the kernel memory management system performs a *MpPullIn* message transaction. It sends, using *ipcCall*(2K) a *MpPullIn* request message to the mapper managing the segment. The mapper is identified by the *ui* field of the segment capability.

The request message consists of an annex (no body) whose head matches the *KnMpIn* structure defined above. The *service* field of this structure is set to KN_MPPULLIN

The kernel demands *dataSize* bytes of data to be updated with the starting offset specified by *dataOffset*,from the segment specified by *segkey* on the invoked mapper, in the local cache specified by *lccap*. The kernel also specifies in the message that the update of *requiredDataSize* bytes of data with the *requiredDataOffset* starting offset is mandatory. The kernel guarantees that *[dataOffset,dataOffset + dataSize]* includes *[requiredDataOffset, requiredDataOffset + requiredDataSize]*. The *dataOffset, dataSize, requiredDataOffset* and *requiredDataSize* parameters are always page aligned. The *pullInId* identifier is the *MpPullIn* message transaction identifier used in *lcPushData(2SEG)*.

The mapper must reply (using *ipcReply*(2K)) with a message whose annex head matches the *KnMpInReply* structure, and whose body (if any) contains the data to be put in the local cache.

The *diag* field is the operation return code. The *diag* field must be either 0 (K_OK), or a negative number. If a negative error code is returned, the kernel returns it to the original kernel call.

The *returnDataSize* field specifies the size of the data to be put in the local cache, and the *returnDataOffset* field specifies the starting offset of the data within the segment. The mapper has to guarantee that *[returnDataOffset, returnDataOffset + returnDataSize]* is included in *[dataOffset, dataOffset + dataSize]* but includes *[requiredDataOffset, requiredDataOffset + requiredDataSize]*. Both *returnDataOffset* and *returnDataSize* must be page aligned.

The size of the body may be less or equal to the *returnDataSize* value. If the body size is less than *returnDataSize*, the memory management system will fill the remaining data with default values: zero or scratch. The default value is zero if the mapper sets the K_FILLZERO flag in the *grantedAccess* field.

If the K_DIRTY flag is set in the *grantedAccess* field, the kernel considers the returned data as dirty: the kernel will later perform an *MpPushOut* operation for the data even if the data is not modified on the site.

If *accessSize* is not zero, the *MpPullIn* data request is combined with a *getAcceess* access request. The mapper must reply to the access request as described in the *MpGetAccess (2SEG)* manual.

In the case of a combined request, the mapper can defer data return until a subsequent non-combined (pure) *MpPullIn* request. To reply without data, the mapper sets the *returnDataSize* field and the reply body size to zero.

If the mapper replies to a combined *MpPullIn* request with data, it can't use the K_DIRTY and K_ORDERED flags together.

ATTRIBUTES

See `attributes`(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

SEE ALSO

`MpGetAccess`(2SEG), `MpPushOut`(2SEG), `rgnMap`(2SEG), `lcOpen`(2SEG), `lcPushData`(2SEG)

NAME

MpPushOut – write data back to mapper

SYNOPSIS

```
#include <mem/chMem.h>
#include <mem/chMapper.h>
MpPushOut
request annex (KnMpOut structure) :
   int           service
   KnKey         segkey
   KnCap         lccap
   VmOffset      dataOffset
   VmSize        dataSize
   VmFlags       options
   KnAsynIoDesc  asynIoDesc
request body:
   unsigned char data [bodysize]
response annex (KnMpOutReply structure) :
   int           diag
```

FEATURES

MEM_VM

DESCRIPTION

Caution - This system call is strictly reserved for internal use only. It MUST NOT be used by any application.

In order to write back the contents of a range of a local cache to the corresponding segment, the kernel memory management system performs a *MpPushOut* message transaction. It sends, using *ipcCall*(2K) an *MpPushOut* request message to the mapper managing the segment. The mapper is identified by the *ui* field of the segment capability.

The request message consists of an annex whose head matches the *KnMpOut* structure defined above, and a body containing the data to be written. The *service* field is set to KN_MPPUSHOUT.

Upon receipt of such a request, the mapper writes data back to the segment specified by *segkey* from the local cache specified by *lccap*. If the segment was created using *MpCreate*(2SEG), the *lccap* argument is undefined.

The *dataOffset* field specifies the starting offset of the data in the segment, and the *dataSize* field the data size. Currently, the *dataSize* field is always equal to the message body size and "fragment"-aligned. The size of the fragment is implementation—dependent and usually equal to the virtual page size divided by 8 (number of bits in one byte). The *dataOffset* field is always page-aligned.

If the K_ASYN_REQUEST flag is set in the *options* field, the mapper can perform an asynchronous write (see below). In this case the kernel fills the *asynIoDesc* structure which has to be passed as a parameter for a subsequent *ioDone* call. The *asynIoDesc* structure is opaque to the mapper.

The mapper must reply to a message whose annex head matches the *KnMpOutReply* structure, where the *diag* field is the operation return code. The *diag* field must be 0 (K_OK) or 1 (K_ASYN_REPLY) or a negative number. If a negative error code is returned, the kernel returns it to the original kernel call (if any). The K_ASYN_REPLY can be returned by the mapper if, and only if, the K_ASYN_REQUEST flag was set by the kernel.

If K_ASYN_REPLY is returned (the mapper performs an asynchronous write), the mapper must invoke an *ioDone* function in order to notify that a write operation has been peformed, as follows:

```
void ioDone (diag, asynIoDesc)
      int             diag ;
      KnAsynIoDesc*   asynIoDesc ;
```

The *diag* parameter must be either 0 (K_OK) or a negative number. If a negative error code is returned, the kernel returns it to the original kernel call (if any).

The *asynIoDesc* parameter points to the *KnAsynIoDesc* structure corresponding to the *MpPushOut* request.

The *ioDone* function can be called asynchronously (in respect of *ipcCall* reply) from base or interrupt level.

RESTRICTIONS

The current kernel implementation never performs asynchronous *MpPushOut* requests to a remote mapper.

The *ioDone* function implementation is only for trusted supervisor actors: an attempt to pass an invalid *asynIoDesc* may produce undefined system behavior.

ATTRIBUTES

See `attributes`(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

SEE ALSO

`MpPullIn`(2SEG), `MpCreate`(2SEG), `lcFlush`(2SEG), `rgnMap`(2SEG), `lcOpen`(2SEG)

NAME

MpRelease – release a temporary segment or notify a local cache destruction

SYNOPSIS

```
#include <mem/chMem.h>
#include <mem/chMapper.h>
MpRelease
request annex (KnMpRelease structure) :
   int     service ;
   KnKey   segkey ;
   KnCap   lccap ;
response annex (KnMpReleaseReply structure) :
   int     diag ;
```

FEATURES

MEM_VM

DESCRIPTION

Caution - This system call is strictly reserved for internal use only. It MUST NOT be used by any application.

When a temporary segment previously created using *MpCreate*(2SEG) is no longer in use, an *MpRelease* message transaction is performed by the kernel. The kernel also performs this request, when destroying a local cache for which the K_RELEASE flag was set by the mapper (see *MpGetAccess*(2SEG)). This request message is sent to the mapper managing the segment, using *ipcCall*(2K). The mapper is identified by the *ui* field of the segment capability.

The request message consists of an annex (no body) whose head matches the *KnMpRelease* structure defined above.

The *service* field is set to KN_MPRELEASE.

The *segkey* field identifies the segment and the *lccap* field, if needed, identifies the local cache just destroyed.

The mapper must reply with a message whose annex head matches the *KnMpReleaseReply* structure, where the *diag* field is the operation return code. The *diag* must be either 0 (K_OK) or a negative number. If a negative error code is returned, the kernel returns it to the original kernel call (if any).

ATTRIBUTES

See `attributes`(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

SEE ALSO

`MpCreate`(2SEG), `MpGetAccess`(2SEG)

NAME

pageIoDone – Inform nucleus I/O is complete on a page list

SYNOPSIS

```
#include <mem/chMem.h>
void pageIoDone(KnPage *page, int diag);
```

FEATURES

PXM_EXT

DESCRIPTION

Caution - This system call is strictly reserved for internal use only. It MUST NOT be used by any application.

The *pageIoDone*function must be invoked by the Proxy-Mapper when a write operation previously triggered by a *pushOutAsyn* up-call is done, or when a read operation following a *dcGetPages* is done. The `page` argument specifies a page list which may be a sub-list of the original lists passed to *pushOutAsyn* or returned to *dcGetPages*. The Proxy-Mapper may split the original lists into a number ofsub-lists and perform the *pageIoDone* call for each sub-list independently. The *diag* argument specifies the result of the operation. If successful it must be set to K_OK, otherwise it must be an error code.

Programmers should take care to invoke the *pageUnmap* nucleus call before invoking the *pageIoDone* call. Doing it the other way round is likely to lead to unpredictable behaviour of the system.

Sub-lists passed to *pageUnmap* and to *pageIoDone* do not need to be identical.

RESTRICTIONS

The current implementation is only applicable to trusted supervisor actors.

RETURN VALUE

NONE

ERRORS

No error messages are returned.

ATTRIBUTES

See `attributes`(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

SEE ALSO

`pageMap`(2SEG), `PxmPushOutAsyn`(2SEG), `dcGetPages`(2SEG)

NAME

pageMap, pageUnmap – Map a list of pages in the current actor address space; Unmap a list of pages

SYNOPSIS

#include <mem/chMem.h>
int **pageMap**(KnPage * *page*, VmFlags *flags*, VmAddr * *addr*, VmSize * *size*, VmOffset * *offset*);

void **pageUnmap**(KnPage * *page*);

FEATURES

PXM_EXT

DESCRIPTION

Caution - This system call is strictly reserved for internal use only. It MUST NOT be used by any application.

The *pageMap* call contiguously maps the page list specified by the `page` argument in the supervisor address space The *flags* argument consists of two parts:

- The K_KNPAGE_TYPE bit field which specifies the type of page list:

`K_KNPAGE_IN` For *getAcc* , *pullIn* or *dcGetPages* page lists

`K_KNPAGE_OUT` For *pushOutAsyn* page lists

- The K_NOWAITFORMEMORY flag; if this flag is set, the call will return immediately with the *K_ENOMEM* error if there is no virtual address space available. If not set, it will wait for virtual memory to become available. This flag must be set if the *pageMap* call is invoked from interrupt level.

The *pageMap* call sets the *addr* output argument to the virtual address to which the page list is mapped. It also sets the `size` output argument to the size of the data, and the *offset* output argument to the offset of the first page of the list. In the case of a *pullIn* up-call, or of a *dcGetPages*, the output argument `size` is set to the number of pages in the list multiplied by the page size. In the case of a *pushOutAsyn* up-call, *pageMap* gives a valid data size which is less than or equal to the number of pages of the list mutiplied by the page size (the last page of the list may be partially valid).

The *pageUnmap* nucleus call unmaps a list of pages specified by the `page` argument which were previously mapped through the *pageMap* call.

It should be noted that the list of pages may be freely split and joined, either before invoking *pageMap* or *pageUnmap* . The lists provided are always sorted incrementally by offset.

RESTRICTIONS

The current implementation is only applicable to trusted supervisor actors.

RETURN VALUE

If successful, K_OK is returned, otherwise an error code is returned.

ERRORS

[K_ENOMEM] The system is out of resources.

ATTRIBUTES

See `attributes`(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

SEE ALSO

`PxmGetAcc`(2SEG) , `dcGetPages`(2SEG) , `pageIoDone`(2SEG)

NAME

pagePhysAddr – Get the physical address of a page

SYNOPSIS

#include <mem/chMem.h>
PhAddr **pagePhysAddr**(KnPage **page*, VmFlags *flags*, VmSize **size*, VmOffset **offset*);

FEATURES

PXM_EXT

DESCRIPTION

Caution - This system call is strictly reserved for internal use only. It MUST NOT be used by any application.

The *pagePhysAddr* call returns the physical address of the physcial page specified by the `page` argument. The *flags* argument specifies the type of page:

K_KNPAGE_IN	Page is used for a *getAcc*, *pullIn* or *dcGetPages* call.
K_KNPAGE_OUT	Page is used for a *pushOutAsyn* call.

When *flags* is set to K_KNPAGE_IN, *pagePhysAddr* sets the `size` output argument to the page size. When *flags* is set to K_KNPAGE_OUT, *pagePhysAddr* gives the valid data size which is less than or equal to the page size. The *pagePhysAddr* call sets the *offset* output argument to the page offset.

RESTRICTIONS

The current implementation is only applicable to trusted supervisor actors.

RETURN VALUE

This call returns the physical address of the page.

ERRORS

none

ATTRIBUTES

See `attributes`(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

SEE ALSO

`pageMap`(2SEG)

NAME

pageSetDirty – Mark a page as dirty

SYNOPSIS

```
#include <mem/chMem.h>
void pageSetDirty(KnPage *page);
```

FEATURES

PXM_EXT

DESCRIPTION

Caution - This system call is strictly reserved for internal use only. It MUST NOT be used by any application.

The *pageSetDirty* call enables the proxy-mapper to mark as dirty a page being read as part of a *pullIn*, *getAcc* up-call or as part of read-ahead processing following a call to *dcGetPages*. This is more flexible than marking a full page list dirty, as a result of a *getAcc* or *pullIn* up-call. Morevover, it also allows the proxy-mapper to mark a page as being dirty during a read-ahead operation following a *dcGetPages* invocation.

RESTRICTIONS

The current implementation is only applicable to trusted supervisor actors.

RETURN VALUE

NONE

ERRORS

No error messages are returned.

ATTRIBUTES

See attributes(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

SEE ALSO

pageMap(2SEG), PxmPushOutAsyn(2SEG), dcGetPages(2SEG)

NAME

pageSgId – Get the segment identifier associated with a page

SYNOPSIS

```
#include <mem/chMem.h>
int pageSgId(KnPage *page);
```

FEATURES

PXM_EXT

DESCRIPTION

Caution - This system call is strictly reserved for internal use only. It MUST NOT be used by any application.

The *pageSgId* call returns the segment identifier corresponding to the data cache to which the page (or page list) specified by the `page` argument is attached. This segment identifier is the one defined by the proxy-mapper at *dcAlloc(2SEG)* invocation.

Note that if the page list used is the one provided by a *getAcc* up-call, *pageSgId* may return NULL, because conditionally attached pages may have been detached by *dcFillZero* or *dcTrunc*.

RESTRICTIONS

The current implementation is only applicable to trusted supervisor actors.

RETURN VALUE

This call returns the segment identifier, or NULL if there are no pages attached to the data cache.

ERRORS

No error messages are returned.

ATTRIBUTES

See `attributes`(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

SEE ALSO

`dcAlloc`(2SEG), `dcFlush`(2SEG), `dcFillZero`(2SEG), `dcTrunc`(2SEG), `PxmGetAcc`(2SEG), `pageMap`(2SEG)

NAME

pageMap, pageUnmap – Map a list of pages in the current actor address space; Unmap a list of pages

SYNOPSIS

#include <mem/chMem.h>
int **pageMap**(KnPage * *page*, VmFlags *flags*, VmAddr * *addr*, VmSize * *size*, VmOffset * *offset*);

void **pageUnmap**(KnPage * *page*);

FEATURES

PXM_EXT

DESCRIPTION

Caution - This system call is strictly reserved for internal use only. It MUST NOT be used by any application.

The *pageMap* call contiguously maps the page list specified by the `page` argument in the supervisor address space The *flags* argument consists of two parts:

- The K_KNPAGE_TYPE bit field which specifies the type of page list:

`K_KNPAGE_IN` For *getAcc* , *pullIn* or *dcGetPages* page lists

`K_KNPAGE_OUT` For *pushOutAsyn* page lists

- The K_NOWAITFORMEMORY flag; if this flag is set, the call will return immediately with the *K_ENOMEM* error if there is no virtual address space available. If not set, it will wait for virtual memory to become available. This flag must be set if the *pageMap* call is invoked from interrupt level.

The *pageMap* call sets the *addr* output argument to the virtual address to which the page list is mapped. It also sets the `size` output argument to the size of the data, and the *offset* output argument to the offset of the first page of the list. In the case of a *pullIn* up-call, or of a *dcGetPages*, the output argument `size` is set to the number of pages in the list multiplied by the page size. In the case of a *pushOutAsyn* up-call, *pageMap* gives a valid data size which is less than or equal to the number of pages of the list mutiplied by the page size (the last page of the list may be partially valid).

The *pageUnmap* nucleus call unmaps a list of pages specified by the `page` argument which were previously mapped through the *pageMap* call.

It should be noted that the list of pages may be freely split and joined, either before invoking *pageMap* or *pageUnmap* . The lists provided are always sorted incrementally by offset.

RESTRICTIONS

The current implementation is only applicable to trusted supervisor actors.

RETURN VALUE

If successful, K_OK is returned, otherwise an error code is returned.

ERRORS

[K_ENOMEM] The system is out of resources.

ATTRIBUTES

See `attributes`(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

SEE ALSO

`PxmGetAcc`(2SEG) , `dcGetPages`(2SEG) , `pageIoDone`(2SEG)

NAME

PxmOpen, PxmClose – Open a Data Segment; Close a Data Segment

SYNOPSIS

```
#include <mem/chMem.h>
void PxmOpen(KnPxmOpenArgs * openArg, void * cookie);

void PxmClose(KnPxmCloseArgs * closeArgvoid, void * cookie);
```

FEATURES

PXM_EXT

DESCRIPTION

Caution - This system call is strictly reserved for internal use only. It MUST NOT be used by any application.

The *PxmOpen/PxmClose* routines are provided by the Proxy-Mapper and are invoked as up-calls by the CHORUS VM. They are invoked as *laps* ; their second argument is the lap cookie as defined at *svLapCreate* time. The *PxmOpen/PxmClose* up-calls are invoked by the VM in order to inform the Proxy-Mapper that the data segment object is currently being used by the VM. The Proxy-Mapper must not destroy an open data segment.

The *KnPxmOpenArgs* data structure has the following members:

```
KnDtPxMapper*   pxm ;
KnSgId          sgId ;
int             write ;
```

The *KnPxmCloseArgs* data structure has exactly the same members. The *pxm* field is a pointer to the *KnExtPxMapper* structure previously defined by a call to *dcPxmDeclare* . It is also associated with the data segment defined by the *sgId* member of the structure at *dcAlloc* time. If the `write` field is set to 0, the data cache will not be modified.If the value is not 0, it may be modified. This allows the Proxy-Mapper to optimize sync or flush operations.

The *PxmOpen* up-call is invoked by the VM in the following cases:

- When *rgnMapFromDtCache* or *rgnInitFromDtCache* are invoked
- When *rgnMapFromActor* or *rgnInitFromActor* or *rgnDup* are invoked, and the source region was created through a call to *rgnMapFromDtCache*
- When a physical page attached to the data cache will be destroyed by the swapper.

The *PxmClose* up-call is invoked by the VM in the following cases:

- When a region mapping the data segment identified by the *sgId* field of the *KnPxmCloseArgs* data structure is destroyed
- When a physical page attached to the corresponding data cache has been destroyed

RESTRICTIONS The current implementation is only applicable to trusted supervisor actors.

RETURN VALUE No error messages are returned.

ERRORS none

ATTRIBUTES See `attributes`(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

SEE ALSO `dcPxmDeclare`(2SEG) , `dcAlloc`(2SEG) , `dcFree`(2SEG) , `svLapCreate`(2K) , `lapInvoke`(2K)

NAME

PxmGetAcc – Get access rights and data on a part of a data segment

SYNOPSIS

#include <mem/chMem.h>
void **PxmGetAcc**(KnPxmGetAccArgs **getArg*, void **cookie*);

FEATURES

PXM_EXT

DESCRIPTION

Caution - This system call is strictly reserved for internal use only. It MUST NOT be used by any application.

The *PxmGetAcc* routine is provided by the Proxy-Mapper and is invoked as an up-call by the CHORUS VM. It is invoked as a *lap*; its second argument is the lap cookie as defined at *svLapCreate* time. The *PxmGetAcc* up-call is invoked by the VM in order to obtain access rights needed to perform an operation on the data cache.

The *KnPxmGetAccArgs* data structure has the following members:

```
KnDtPxMapper*   pxm ;
KnSgId          sgId ;
KnPxmAcc*       accParms ;
KnPage*         page ;
VmSize*         size ;
```

The *pxm* field is a pointer to the *KnExtPxMapper* structure previously defined by a call to *dcPxmDeclare* and associated with the data segment defined by the *sgId* member of the structure at *dcAlloc* time.

The access rights are specified by the data structure pointed to by the *accParms* field. The *KnPxmAcc* structure contains the following fields:

```
KnPxmAccReq   req ;
KnPxmAccRep   rep ;
void*         lock ;
```

The *req* field describes the access rights requested by the VM, while the *rep* field is filled by the Proxy-Mapper to define the actual access rights returned to the VM. The *KnPxmAccReq* structure contains the following members:

```
VmFlags   access ;
VmOffset  start ;
VmOffset  end ;
VmOffset  reqStart ;
VmOffset  reqEnd ;
```

The *access* field specifies the access rights, and may be set to either K_READABLE or K_WRITABLE. When set to K_READABLE, the corresponding data will be read—only by the VM. If set to K_WRITABLE, the corresponding data may be modified by the VM. In such a case, the Proxy-Mapper should insure that

backing store (if needed) will be available when data is to be pushed out. The *start* and *end* fields specify the access range desired, the *reqStart* and *reqEnd* specify the access range required.

The *KnPxmAccRep* structure contains the following members:

```
VmFlags   access ;
VmOffset  start ;
VmOffset  end ;
int       diag ;
```

where the *access* field specifies the access rights returned by the Proxy-Mapper to the VM (either K_READABLE or K_WRITABLE). The *start* and *end* fields specify the range of bytes for which the Proxy-Mapper grants access rights to the VM as a result of the *PxmGetAcc* up-call. The *diag* field contains the diagnostic returned by the Proxy-Mapper to the VM. If set to K_OK, the up-call has been successful and the fields returned are meaningful, otherwise it is set to the appropriate error code. If the error code returned is K_EBUSY, the VM will invoke the *PxmGetAcc* up-call again.

The Proxy-Mapper could grant write access in response to a read-only access request, but it must grant write access in response to a write access request. The Proxy-Mapper must guarantee that the [*rep.start*, *rep.end*] range is included in the [*req.start*, *req.end*] range, but also includes the [*req.reqStart*, *req.reqEnd*] range. . The *rep.start* and *rep.end*+1 values must be fragment-aligned.

The VM may request data at the same time as access rights. In this case, the `page` argument specifies the list of pages which should be read from the data segment. The `page` field points to a *KnPage* structure which specifies the first page of the list. The *KnPage* structure has the following members:

```
struct KnPage*   next ;
void*            work ;
```

The Proxy-Mapper must set the `size` output field to the size of valid data (from the beginning of the page list). If no data are requested, the `page` is NULL and the Proxy-Mapper must not refer to the `size` field. The *KnPage* structure is the physical page descriptor exported by the VM. The *next* field points to the next page of the list and is set to NULL for the last page of the list. The *work* field is not used by the VM while the page is owned by the Proxy-Mapper, it may be used by the Proxy-Mapper while it owns the page.

There are two VM operations which can invoke the *PxmGetAcc* up-call:

- *dcRead/dcWrite* invocations
- A page fault on a region created by *rgnMapFromDtCache.*

In the case of *dcRead* or *dcWrite*, the *lock* field contains the *accLock* argument of the *dcRead/dcWrite* call. Thus, the *lock* field is an input argument when *PxmGetAcc* is called as a result of *dcRead* or *dcWrite*.

In the case of a page fault, the *lock* field is set to NULL and the K_PAGEFAULT bit is set in the *req.access* field. If the *PxmGetAcc* is performed successfully, the VM invokes the *PxmRelAccLock* up-call passing the value of the *lock* field as an arguemtn. This value is returned to the VM by the Proxy-Mapper at the completion of the *PxmGetAcc* up-call. Thus, in the case of page fault, the *lock* field is an input/output argument of the *PxmGetAcc* up-call. Note that the *relAccLock* up-call is never invoked as a result of a *dcRead* or *dcWrite* nucleus call.

The pages of the *getArg->page* list are conditionally attached to the data cache. It means that they are not visible to *dcFlush* and therefore *dcFlush* cannot be blocked on them. If *dcFillZero* or *dcTrunc* find a conditionally attached page (which must be filled) they perform the following actions:

- Detach this page from the data cache
- Allocate a new physical page and attach it to the data cache
- Invoke the *PxmPullIn* up-call to read data from the data segment, if needed
- Fill the appropriate part of the page with zero, if needed

Once access rights are obtained, the VM attaches the pages to the data cache fully (they become visible to *dcFlush*) and invokes the *PxmRrelAccLock* up-call(if needed, for example in the case of a page fault). Note that, in the case of a pure *PxmGetAcc*, the *getArg->page* is NULL The *PxmRelAccLock* up-call is invoked when the access rights of the page have already been upgraded.

RESTRICTIONS

The current implementation is done only for trusted supervisor actors.

RETURN VALUE

None.

ERRORS

none

ATTRIBUTES

See `attributes`(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

SEE ALSO

`dcPxmDeclare`(2SEG), `dcAlloc`(2SEG), `svLapCreate`(2K), `lapInvoke`(2K), `pageMap`(2SEG), `dcRead`(2SEG), `dcWrite`(2SEG), `rgnMapFromDtCache`(2SEG), `PxmRelAccLock`(2SEG)

NAME

PxmOpen, PxmClose – Open a Data Segment; Close a Data Segment

SYNOPSIS

#include <mem/chMem.h>
void **PxmOpen**(KnPxmOpenArgs * *openArg*, void * *cookie*);

void **PxmClose**(KnPxmCloseArgs * *closeArgvoid*, void * *cookie*);

FEATURES

PXM_EXT

DESCRIPTION

Caution - This system call is strictly reserved for internal use only. It MUST NOT be used by any application.

The *PxmOpen/PxmClose* routines are provided by the Proxy-Mapper and are invoked as up-calls by the CHORUS VM. They are invoked as *laps* ; their second argument is the lap cookie as defined at *svLapCreate* time. The *PxmOpen/PxmClose* up-calls are invoked by the VM in order to inform the Proxy-Mapper that the data segment object is currently being used by the VM. The Proxy-Mapper must not destroy an open data segment.

The *KnPxmOpenArgs* data structure has the following members:

```
KnDtPxMapper*   pxm ;
KnSgId          sgId ;
int             write ;
```

The *KnPxmCloseArgs* data structure has exactly the same members. The *pxm* field is a pointer to the *KnExtPxMapper* structure previously defined by a call to *dcPxmDeclare* . It is also associated with the data segment defined by the *sgId* member of the structure at *dcAlloc* time. If the `write` field is set to 0, the data cache will not be modified.If the value is not 0, it may be modified. This allows the Proxy-Mapper to optimize sync or flush operations.

The *PxmOpen* up-call is invoked by the VM in the following cases:

- When *rgnMapFromDtCache* or *rgnInitFromDtCache* are invoked
- When *rgnMapFromActor* or *rgnInitFromActor* or *rgnDup* are invoked, and the source region was created through a call to *rgnMapFromDtCache*
- When a physical page attached to the data cache will be destroyed by the swapper.

The *PxmClose* up-call is invoked by the VM in the following cases:

- When a region mapping the data segment identified by the *sgId* field of the *KnPxmCloseArgs* data structure is destroyed
- When a physical page attached to the corresponding data cache has been destroyed

RESTRICTIONS The current implementation is only applicable to trusted supervisor actors.

RETURN VALUE No error messages are returned.

ERRORS none

ATTRIBUTES See `attributes`(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

SEE ALSO `dcPxmDeclare`(2SEG) , `dcAlloc`(2SEG) , `dcFree`(2SEG) , `svLapCreate`(2K) , `lapInvoke`(2K)

NAME

PxmPullIn – Read data from a data segment

SYNOPSIS

```
#include <mem/chMem.h>
void PxmPullIn(KnPxmPullInArgs *pullArg, void *cookie);
```

FEATURES

PXM_EXT

DESCRIPTION

Caution - This system call is strictly reserved for internal use only. It MUST NOT be used by any application.

The *PxmPullIn* routine is provided by the Proxy-Mapper and is invoked as an up-call by the CHORUS VM. It is invoked as a *lap*; its second argument is the lap cookie as defined at *svLapCreate* time. The *PxmPullIn* up-call is invoked by the VM in order to read data from the data segment into the data cache. The *KnPxmPullInArgs* structure has the following members:

```
KnDtPxMapper*   pxm ;
KnSgId          sgId ;
KnPage*         page ;
```

The *sgId* field specifies the corresponding data segment. The page field field specifies the list of pages to be read. It points to a *KnPage* structure (see *PxmGetAcc(2SEG)*) which specifies the first page of the list.

The pages of the *PxmPullIn* page list are fully attached to the data cache and write locked (exclusively). This means that all Nucleus calls (including *dcFlush*, *dcFillZero* and *dcTrunc*) which acquire a lock on this type of page are blocked until *pullIn* is finished.

The VM calls *PxmPullIn* when access rights have already been obtained by a previous *PxmGetAcc*. In the case of page fault, the *PxmRelAccLock* (see the *PxmGetAcc(2SEG)* man page) up-call is invoked after *PxmPullIn*.

RESTRICTIONS

The current implementation is only applicable to trusted supervisor actors.

RETURN VALUE

None.

ATTRIBUTES

See attributes(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

SEE ALSO

dcPxmDeclare(2SEG), dcAlloc(2SEG), dcFree(2SEG), svLapCreate(2K), lapInvoke(2K), PxmGetAcc(2SEG), PxmRelAccLock(2SEG), pageMap(2SEG)

NAME

PxmPushOutAsyn – Write asynchronously to a data segment

SYNOPSIS

```
#include <mem/chMem.h>
void PxmPushOutAsyn(KnPxmPushOutArgs *pushArg, void *cookie);
```

FEATURES

PXM_EXT

DESCRIPTION

Caution - This system call is strictly reserved for internal use only. It MUST NOT be used by any application.

The *PxmPushOutAsyn* routine is provided by the Proxy-Mapper and is invoked as an up-call by the CHORUS VM. It is invoked as a *lap*; its second argument is the lap cookie as defined at *svLapCreate* time. The *PxmPushOutAsyn* up-call is invoked by the VM in order to write data to a data segment from the data cache. The *KnPxmPushOutArgs* structure has the following members:

```
KnDtPxMapper*   pxm ;
KnSgId          sgId ;
KnPage*         page ;
VmFlags         flags ;
void*           pdDesc ;
```

The *sgId* field specifies the corresponding data segment. The page field specifies the list of pages to be written. It points to a *KnPage* structure (see *PxmGetAcc(2SEG)*) which specifies the first page of the list.

If at least one page of the list has been modified, The VM sets the K_DIRTY bit in the *flags* field. The *flags* field also contains the K_POUT_TYPE bit field which specifies the reason for the *PxmPushOutAsyn* up-call:

K_POUT_FLUSH	This up-call is the result of a call to *dcFlush*.
K_POUT_SYNC	This up-call is the result of a call to *dcSync*.
K_POUT_SWAP	This up-call is invoked by the swapper.

If *PxmPushOutAsyn* is invoked by *dcFlush*, the *pdDesc* field is the *pout* argument of the *dcFlush* nucleus call, otherwise it is set to NULL.

The Proxy-Mapper must return from the *PxmPushOutAsyn* up-call and must perform the *pageIoDone* nucleus call after the write operation is completed. This may be done either before or after returning from the *PxmPushOutAsyn* up-call. There is no need for the Proxy-Mapper to invoke the *pageIoDone* call with the same page list. Multiple calls to *pageIoDone* may be performed one with disjointed sub-lists of pages.

The pages of the page list are fully attached to the data cache and locked. In the case of *dcFlush* or swap, the pages are write locked (exclusively). In the case of *dcSync*, the pages are read locked (share). Note that in order to avoid a

concurrent *PxmPushOutAsyn* of the same page, *dcSync* first acquires the write lock, and then downgrades it to the read one. This implies that all Nucleus calls (including *dcFlush*, *dcFillZero* and *dcTrunc*) which acquire a lock on this type of page are blocked until *pageIoDone* is finished.

RESTRICTIONS

The current implementation is only applicable to trusted supervisor actors.

RETURN VALUE

No error messages are returned.

ERRORS

none

ATTRIBUTES

See `attributes`(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

SEE ALSO

`dcPxmDeclare`(2SEG), `dcAlloc`(2SEG), `dcFree`(2SEG), `svLapCreate`(2K), `lapInvoke`(2K), `PxmGetAcc`(2SEG), `PxmRelAccLock`(2SEG), `pageMap`(2SEG), `pageIoDone`(2SEG)

NAME

PxmRelAccLock – Release an access lock

SYNOPSIS

#include <mem/chMem.h>
void **PxmRelAccLock**(KnPxmRelAccArgs **relArg*, void **cookie*);

FEATURES

PXM_EXT

DESCRIPTION

Caution - This system call is strictly reserved for internal use only. It MUST NOT be used by any application.

The *PxmRelAccLock* routine is provided by the Proxy-Mapper and is invoked as an up-call by the CHORUS VM. It is invoked as a *lap*; , its second argument is the lap cookie as defined at *svLapCreate* time. The *PxmRelAccLock* up-call is invoked by the VM after a previous up-call to *PxmGetAcc* resulting from a page fault. In such a case, the Proxy-Mapper must set the output *accLock* field. When the page fault has been fully processed by the VM, the VM invokes the *PxmRelAccLock* up-call.

The *KnPxmRelAccArgs* structure has the following members:

```
KnDtPxMapper*   pxm ;
KnSgId          sgId ;
void*           lock ;
```

The *sgId* identifies the data segment on which the previous *PxmGetAcc* up-call was performed. The *lock* field is set to the value returned by the *PxmGetAcc* up-call.

RESTRICTIONS

The current implementation is only applicable to trusted supervisor actors.

RETURN VALUE

None.

ERRORS

No error messasges are returned.

ATTRIBUTES

See `attributes`(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

SEE ALSO

`dcPxmDeclare`(2SEG), `dcAlloc`(2SEG), `dcFree`(2SEG), `svLapCreate`(2K), `lapInvoke`(2K), `dcCluster`(2SEG)

NAME

PxmStat – Get information about a data segment

SYNOPSIS

```
#include <mem/chMem.h>
void PxmStat(KnPxmStatArgs *statArg, void *cookie);
```

FEATURES

PXM_EXT

DESCRIPTION

Caution - This system call is strictly reserved for internal use only. It MUST NOT be used by any application.

The *PxmStat* routine is provided by the Proxy-Mapper and is invoked as an up-call by the CHORUS VM. It is invoked as a *lap*; its second argument is the lap cookie as defined at *svLapCreate* time. The *PxmStat* up-call is invoked by the VM in order to obtain data segment—specific information as part of a call to *rgnStat*. The *KnPxmStatArgs* structure is composed of the following fields:

```
KnDtPxMapper*   pxm ;
KnSgId          sgId ;
char*           buff ;
int             size ;
```

The *sgId* field specifies the corresponding data segment. The *buff* field points to the buffer to which the information is to be copied. The `size` field specifies the buffer size in bytes. If the buffer size is sufficient for the data segment information to be copied, the Proxy-Mapper copies them to the buffer, otherwise, it does nothing. The *PxmStat* funciton returns the size of the data segment information.

RESTRICTIONS

The current implementation is only applicable to trusted supervisor actors.

RETURN VALUE

If successful, *PxmStat* returns the size of the data segement information in bytes.

ERRORS

None.

ATTRIBUTES

See `attributes`(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

SEE ALSO

`dcPxmDeclare`(2SEG), `dcAlloc`(2SEG), `dcFree`(2SEG), `svLapCreate`(2K), `lapInvoke`(2K), `rgnStat`(2K)

NAME

PxmSwapOut – Customize next swap out

SYNOPSIS

#include <mem/chMem.h>
void **PxmSwapOut**(KnPxmSwapOutArgs **swArg*, void **cookie*);

FEATURES

PXM_EXT

DESCRIPTION

Caution - This system call is strictly reserved for internal use only. It MUST NOT be used by any application.

The *PxmSwapOut* routine is provided by the Proxy-Mapper and is invoked as an up-call by the CHORUS VM. It is invoked as a *lap*; its second argument is the lap cookie as defined at *svLapCreate* time. The *PxmSwapOut* up-call is invoked by the VM in order to inform the Proxy-Mapper that a page is going to be pushed out (due to action by the VM daemons), as opposed to a push out resulting from a *dcFlush* or a *dcSync*. This up-call is an opportunity for the Proxy-Mapper to tailor the behavior of the VM, so that swap out will occur on ranges that suit the Proxy-Mapper's needs.

The *KnPxmSwapOutArgs* data structure has the following fields:

```
KnDtPxMapper*   pxm ;
KnSgId          sgId ;
KnPage*         page ;
VmOffset*       offset ;
VmOffset*       end ;
```

The *pxm* field points to the Proxy-Mapper definition associated with the segment at the time of *dcAlloc*. The *sgId* field specifies the corresponding data segment. The page field points to the list of pages which are going to be pushed out by the VM. The starting offset of this contiguous list of pages is provided at the location pointed to by the *offset* field. The last offset of the range to be swapped out is defined as the location pointed to by the *end* field. The *PxmSwapOut* up-call enables the Proxy-Mapper to change the range of pages to be swapped out, by changing the values pointed to by the input/output fields: *offset* and *end*. If no *PxmSwapOut* up-call routine has been defined in the *pxm* structure associated with the data segment at *dcAlloc* time, swap out will be performed by the VM according to its own rules based on the cluster size (see *dcCluster*).

RESTRICTIONS

The current implementation is only applicable to trusted supervisor actors.

RETURN VALUE

None.

ERRORS

No error codes are returned.

ATTRIBUTES

See attributes(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

SEE ALSO `dcPxmDeclare`(2SEG), `dcAlloc`(2SEG), `dcFree`(2SEG), `svLapCreate`(2K), `lapInvoke`(2K), `dcCluster`(2SEG)

NAME

rgnFlush – Flush a region

SYNOPSIS

```
#include <mem/chMem.h>
int rgnFlush(KnCap *actorcap, VmOffset start, VmSize size, VmFlags flags);
```

FEATURES

PXM_EXT

DESCRIPTION

Caution - This system call is strictly reserved for internal use only. It MUST NOT be used by any application.

The *rgnFlush* call performs a flush operation on the ranges of data caches mapped to an address range of an actor address space. The address range must be page-aligned.

The target actor is specified by *actorcap* - a pointer to the actor capability. If *actorcap* is K_MYACTOR, the address space of the current actor is used. If *actorcap* is K_SVACTOR, the supervisor address space is used.

The *flags* argument specifies the way the flush is performed as well as the upper access rights for the target parts after the flush operation.

If the K_COPYBACK flag is set, *rgnFlush* writes back all (even clean) cached data of the target parts and does not change the parts' access rights.

If the K_WRITABLE flag is set, *rgnFlush* writes back all modified data of the target parts and does not change the parts' access rights.

If the K_READABLE flag is set, *rgnFlush* writes back all modified data of the target parts and sets the parts' access rights to read only.

If the K_FREEZE flag is set, *rgnFlush* writes back all modified data of the target parts and sets the parts to non-accessible.

If the K_NOACCESS flag is set, *rgnFlush* writes back all modified data of the target parts and invalidates the parts.

If the K_DESTROY flag is set, *rgnFlush* invalidates the target parts without writing back any data.

If K_ASYNC flag is set, the vm performs the asynchronous write operations required by *rgnFlush.*

If K_PAGEFAULT flag is set, *rgnFlush* is the result of a page fault. This type of *lcFlush*operation could break the atomicity of an *sgRead/sgWrite* operation running concurrently on the same local cache parts.

RESTRICTIONS

The target actor and the current actor must be located on the same site.

RETURN VALUE

If successful K_OK is returned, otherwise a negative error code is returned.

ERRORS

[K_EFAULT]	Some of the arguments provided are outside the caller's address space.

[K_EINVAL]	An inconsistent actor capability was provided.
[K_EROUND]	*start* or `size` isn't page-aligned.
[K_EADDR]	Some or all the addresses from the target address range are invalid.
[K_EOFFSET]	Tried to flush a segment outside the valid offset range in a segment.
[K_EINVAL]	The *flags* argument contains invalid flag values.
[K_EBUSY]	Tried to invalidate or destroy a no–demand (mapped to a region with the K_NODEMAND attribute) physical memory.

ATTRIBUTES

See `attributes`(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

SEE ALSO

`rgnInitFromDtCache`(2SEG), `rgnAllocate`(2K), `rgnFree`(2K)

NAME

rgnInit – allocate a region in an actor address space and initialize it from a segment

SYNOPSIS

```
#include <mem/chMem.h>
int rgnInit(KnCap *actorcap, KnRgnDesc *rgndesc, KnObjDesc *segdesc);
```

FEATURES

MEM_VM

DESCRIPTION

Caution - This system call is strictly reserved for internal use only. It MUST NOT be used by any application.

The *rgnInit* call creates a region in the address space of the target actor and maps a volatile copy of a segment range to the region. The target actor is specified by *actorcap* - a pointer to the target actor capability. If *actorcap* is K_MYACTOR, the address space of the current actor is used. If *actorcap* is K_SVACTOR, the region is allocated in the supervisor address space and isn't attached to any particular supervisor actor; it cannot therefore be implicitly deallocated by an *actorDelete*(2K) of a supervisor actor.

The *rgndesc* pointer points to a *KnRgnDesc* structure containing the specification of the region to be created, as described in *rgnAllocate (2K)*.

The *segdesc* pointer points to a *KnObjDesc* structure containing the source data specification. The *KnObjDesc* fields are as follows:

```
KnCap     dataObject ;
VmOffset  startOffset ;
VmSize    size ;
```

The *dataObject* field definesthe capability of the source segment.

The *startOffset* defines the starting offset of the data in the segment. Its value must be aligned to a virtual page boundary.

The `size` field defines the size of the data.

The caller can specify any of the *rgndesc->options* as described in *rgnAllocate (2K)*, except that the K_RESERVED flag is prohibited. The K_FILLZERO option is also interpreted differently here: only the range from *rgndesc->startAddr + segdesc->size* to *rgndesc->startAddr + rgndesc->size - 1* will be zero-filled.

The kernel uses the standard Chorus-IPC based kernel-mapper protocol (see *MpPullIn(2SEG), MpGetAccess(2SEG))* to read the current state of the segment prior to copying it to newly allocated volatile memory, and mapping the memory to the region created. The kernel implementation can defer the effective data reading and copy.

RETURN VALUE

If successful K_OK is returned, otherwise a negative error code is returned.

ERRORS

[K_EFAULT]	Some of the arguements provided are outside the caller's address space.
[K_EINVAL]	An inconsistent actor capability was provided.
[K_EUNKNOWN]	*actorcap* does not specify a reachable actor.
[K_EROUND]	*rgndesc->startAddr* is not page-aligned.
[K_EROUND]	*segdesc->startOffset* is not page-aligned.
[K_ESPACE]	Tried to create a region outside the valid range for the address space of an actor as returned by *vmStat*(2K).
[K_ESPACE]	*rgndesc->size* is zero.
[K_EOFFSET]	Attempted to map a segment outside its valid offset range as returned by *vmStat*(2K).
[K_ESIZE]	*segdesc->size* is greater than *rgndesc->size*.
[K_EOVERLAP]	The K_ANYWHERE option was specified and there is insufficient room available in the address space to create the region.
[K_EOVERLAP]	The K_RESTRICTIVE option was specified and there is insufficient room in the target address range.
[K_EOVERLAP]	the region created overlaps an existing region.
[K_ENOMEM]	The system is out of resources.
[K_EMAPPER]	The segment mapper doesn't respect the kernel/mapper protocol.

RESTRICTIONS

The target actor and the current actor must be located on the same site.

ATTRIBUTES

See `attributes`(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

SEE ALSO

`rgnAllocate`(2K), `rgnMap`(2SEG), `rgnStat`(2K), `MpPullIn`(2SEG), `vmStat`(2K)

NAME

rgnInitFromDtCache – Create a region in an actor address space and initialize it from a segment

SYNOPSIS

#include <mem/chMem.h>
int **rgnInitFromDtCache**(KnCap **actorcap*, KnRgnDesc **rgndesc*, KnLcId *lcid*, VmOffset *start*, VmOffset *end*);

FEATURES

PXM_EXT

DESCRIPTION

Caution - This system call is strictly reserved for internal use only. It MUST NOT be used by any application.

The *rgnInitFromDtCache* call allocates a region in the actor address space specified by *actorcap*. It initializes from the segment associated with the data cache specified by *lcid*. If *actorcap* is set to `K_MYACTOR`, the current actor is used. If *actorcap* is set to `K_SVACTOR`, the region is allocated in supervisor address space and is not attached to any particular supervisor actor.

The *rgnDesc* pointer points to a *KnRgnDesc* structure containing the specification for the region to be created, as described in *rgnAllocate(2K)*. Note that the `K_FILLZERO` option is interpreted differently here; the range of bytes comprised between *rgndesc->startAddr+end+1* and *rgndesc->startAddr+size-1* will be zero-filled.

The *start* argument specifies which offset of the segment will be mapped to the first location of the newly created region. Its value must be aligned on a virtual page boundary. The *end* argument specifies the offset of the last byte of the data segment to be mapped to the created region.

RESTRICTIONS

The current implementation is only applicable to trusted supervisor actors.

RETURN VALUE

If successful K_OK is returned, otherwise a negative error code is returned.

ERRORS

[K_EFAULT]	Some of the arguments provided are outside the caller's address space.
[K_EINVAL]	An inconsistent actor capability was provided.
[K_ESIZE]	The *end* argument and the `size` field of the *rgndesc* argument are inconsistent.
[K_ENOMEM]	The system is out of resources, or some or all of the memory identified by the operation could not be locked when K_NODEMAND and K_NOWAITFORMEMORY have both been specified.
[K_EOFFSET]	Attempted to map a segment outside its valid offset range.

[K_EOVERLAP]	The K_ANYWHERE option was specified and there is insufficient room in the address space to allocate the region. The K_RETSRICTIVE option was specified and there is insufficient room in the target address range. The region specified overlaps an existing region. The K_RESTRICTIVE option was specified and the right boundary is less than the left one.
[K_EROUND]	The *start* argument is not a multiple of the page size as returned by *vmPageSize(2K).*
[K_ESPACE]	Attempted to allocate a region outside the valid range for the address space of an actor as returned by *vmStat(2K)*

ATTRIBUTES

See `attributes`(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

SEE ALSO

`rgnInitFromDtCache`(2SEG), `rgnAllocate`(2K), `rgnFree`(2K)

NAME

rgnMap – create a region in an actor address space and map a segment

SYNOPSIS

```
#include <mem/chMem.h>
int rgnMap(KnCap *actorcap, KnRgnDesc *rgndesc, KnObjDesc *segdesc);
```

FEATURES

MEM_VM

DESCRIPTION

Caution - This system call is strictly reserved for internal use only. It MUST NOT be used by any application.

The *rgnMap* call creates a region in the address space of the target actor and maps a segment to the region.

The target actor is specified by *actorcap* - a pointer to the target actor capability. If *actorcap* is K_MYACTOR, the address space of the current actor is used. If *actorcap* is K_SVACTOR, the region is allocated in the supervisor address space and isn't attached to any particular supervisor actor; it cannot therefore be implicitly deallocated using an *actorDelete*(2K) of a supervisor actor.

The *rgndesc* pointer points to a *KnRgnDesc* structure containing the specification of the region to be created as described in *rgnAllocate (2K).*

The *segdesc* pointer points to a *KnObjDesc* structure containing the specification of the segment range to be mapped to the created region described in *rgnInit (2SEG).* Note that the `size` field will be ignored because the size of the range mapped is equal to the size of the region created.

The caller can specify any of the *rgndesc->options* as described in *rgnAllocate (2K)* except that the K_FILLZERO flag is ignored, and theK_RESERVED flag is prohibited. Also, the caller can specify K_NOSYNC flag.

If the K_NOSYNC flag is specified, the kernel writes back dirty pages of the segment as late as possible. Otherwise, any modification of a segment is written back within a period of time specified using a system configuration parameter..

RETURN VALUE

If successful K_OK is returned, otherwise a negative error code is returned.

ERRORS

[K_EFAULT]	Some of the arguments provided are outside the caller's address space.
[K_EINVAL]	An inconsistent actor capability was provided.
[K_EUNKNOWN]	*actorcap* does not specify a reachable actor.
[K_EROUND]	*rgndesc->startAddr* is not page-aligned.
[K_EROUND]	*segdesc->startOffset* is not page-aligned.
[K_ESPACE]	Tried to create a region outside the valid range for the address space of an actor as returned by *vmStat*(2K).

[K_ESPACE]	*rgndesc->size* is zero.
[K_EOFFSET]	Attempted to map a segment outside its valid offset range as returned by *vmStat*(2K).
[K_EOVERLAP]	The K_ANYWHERE option was specified and there is insufficient room available in the address space to create the region.
[K_EOVERLAP]	The K_RESTRICTIVE option was specified and there is insufficient room in the target address range.
[K_EOVERLAP]	The region created overlaps an existing region.
[K_ENOMEM]	The system is out of resources.
[K_EMAPPER]	The segment mapper doesn't respect the kernel/mapper protocol.

RESTRICTIONS

The target actor and the current actor must be located on the same site.

ATTRIBUTES

See `attributes`(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

SEE ALSO

`rgnAllocate`(2K), `rgnInit`(2SEG), `rgnStat`(2K), `MpPullIn`(2SEG), `MpGetAccess`(2SEG)

NAME

rgnMapFromDtCache – Create a region in an actor address space and map a segment

SYNOPSIS

```
#include <mem/chMem.h>
int rgnMapFromDtCache(KnCap *actorcap, KnRgnDesc *rgndesc, KnLcId lcid, VmOffset start);
```

FEATURES

PXM_EXT

DESCRIPTION

Caution - This system call is strictly reserved for internal use only. It MUST NOT be used by any application.

The *rgnMapFromDtCache* call creates a region in the actor address space specified by *actorcap*, and maps the segment associated with the data cache specified by *lcid* to it. If *actorcap* is set to K_MYACTOR, the current actor is used. If *actorcap* is set to K_SVACTOR, the region is allocated in the supervisor address space and is not attached to any particular supervisor actor.

The *rgnDesc* pointer points to a *KnRgnDesc* structure containing the specification for the region to be created as described in *rgnAllocate(2K)*. Note that the K_FILLZERO option will be ignored and that the K_RESERVED flag is prohibited.

The *start* argument specifies which offset of the segment will be mapped to the first location of the newly created region. Its value must be aligned on a virtual page boundary.

RESTRICTIONS

The current implementation is only applicable to trusted supervisor actors.

RETURN VALUE

If successful K_OK is returned, otherwise a negative error code is returned.

ERRORS

[K_EFAULT]	Some of the arguments provided are outside the caller's address space. This error will only be returned if the invoker is a user actor.
[K_EINVAL]	An inconsistent actor capability was given.
[K_ENOMEM]	The system is out of resources or some or all of the memory identified by the operation could not be locked when K_NODEMAND and K_NOWAITFORMEMORY were both specified.
[K_EOFFSET]	Attempted to map a segment outside its valid offset range.
[K_EOVERLAP]	The K_ANYWHERE option was specified and there is insufficient room in the address space to allocate the region. The K_RESTRICTIVE option was specified and there is insufficient room in the target address range. The specified region overlaps an existing region. The K_RESTRICTIVE

	option was specified and the right boundary is less than the left one.
[K_EROUND]	The *start* argument is not a multiple of the page size as returned by *vmPageSize(2K).*
[K_ESPACE]	Attempted to allocate a region outside the valid range for the address space of an actor as returned by *vmStat(2K)*

ATTRIBUTES

See `attributes`(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

SEE ALSO

`rgnInitFromDtCache`(2SEG), `rgnAllocate`(2K), `rgnFree`(2K)

NAME

lcFlush, lcSetRights, sgFlush, sgSyncAll, vmFlush – flush local cache(s)

SYNOPSIS

```
#include <mem/chMem.h>
int lcFlush(KnObjDesc * lcdesc, VmFlags flags, unsigned long ordernb);

int lcSetRights(KnObjDesc * lcdesc, VmFlags flags, unsigned long ordernb);

int sgFlush(KnObjDesc * segdesc, VmFlags flags);

int sgSyncAll(void);

int vmFlush(KnCap * actorcap, VmAddr address, VmSize size, VmFlags flags);
```

FEATURES

MEM_VM

DESCRIPTION

Caution - This system call is strictly reserved for internal use only. It MUST NOT be used by any application.

The *lcFlush* call performs a flush operation on a range of a local cache. The *lcdesc* argument points to a *KnObjDesc* structure whose members are the following:

```
KnCap     dataObject ;
VmOffset  startOffset ;
VmSize    size ;
```

where the *dataObject* field is the local cache capability, the *startOffset* field is the range start offset in the local cache, and the `size` field is the range size. Both *startOffset* and `size` must be fragment-aligned. The size of the fragment is implementation–dependent and is usually equal to the virtual page size divided by 8 (number of bits in one byte).

For *lcFlush*, the target local cache is directly specified by its capability; the flush operation is applied even if the cache is managed by a remote site.

The *sgFlush* call takes the same arguments, except the *dataObject* field specifies the capability of the cached segment. For *sgFlush*, the target local cache is indirectly specified by the capability of the corresponding segment; the flush operation implicitly applies to the segment's cache(s) located on the caller's site.

The *vmFlush* call performs a flush operation on the ranges of the local caches mapped to an address range of an actor address space. The address range must be page-aligned.

The target actor is specified by *actorcap* - a pointer to the actor capability. If *actorcap* is K_MYACTOR, the address space of the current actor is used. If *actorcap* is K_SVACTOR, the supervisor address space is used.

The *flags* argument specifies the mode of the flush and the upper access rights for the target parts after the flush operation.

If the K_COPYBACK flag is set, *xxFlush* writes back any (even clean) cached data of the target parts and keeps the parts access rights unchanged.

If the K_WRITABLE flag is set, *xxFlush* writes back all modified data of the target parts and keeps the parts access rights unchanged.

If the K_READABLE flag is set, *xxFlush* writes back all modified data of the target parts and then sets the parts to read access only.

If the K_FREEZE flag is set, *xxFlush* writes back all modified data of the target parts and then sets the parts as non-accessible.

If the K_NOACCESS flag is set, *xxFlush* writes back all modified data of the target parts and invalidates them.

If the K_DESTROY flag is set, *xxFlush* invalidates the target parts without writing back.

If the K_ASYNC flag is set, the vm performs the write operations required by the *xxFlush* asynchronously.

If K_PAGEFAULT flag is set, *xxFlush* is a result of a page fault. This type of *lcFlush* operation could break the atomicity of a sgRead / sgWrite operation running concurrently on the same local cache parts.

The K_ORDERED flag can only be used with an *lcFlush* or an *lcSetRights* request. If the flags is set, the *orderNb* argument specifies the *lcFlush* message order number. It allows the kernel to detect the situation when two mesages sent by a mapper in one order are received by the kernel in another. For instance, if the mapper grants certain access rights in a *MpGetAccess* reply first, then, processing another request, calls an *lcFlush* to recall the access rights, considering the order numbers, the kernel is aware that the access returned by the *MpGetAccess* reply was already recalled by the mapper, and performs another *MpGetAccess* request.

The *lcSetRights* operation is similar to the *lcFlush* operation, except that it only changes data protections without invalidation and / or writing back: the K_WRITABLE flag is ignored, the K_READABLE flag sets the target parts read—only, whereas the K_FREEZE and K_NOACCESS flags set the target parts to non-accessible. The K_COPYBACK, K_ASYNC and K_DESTROY flags are prohibited.

The *sgSyncAll* operation writes back asynchronously all dirty parts of all local caches on the site, except the local caches mapped at least once to a region with the K_NOSYNC attribute (see *rgnMap(2SEG)*).

RETURN VALUE

If successful K_OK is returned, otherwise a negative error code is returned.

ERRORS

[K_EFAULT]	Some of the arguments provided are outside the caller's address space.

[K_EINVAL]	An inconsistent actor capability was provided.
[K_EUNKNOWN]	*actorcap* does not specify a reachable actor.
[K_EROUND]	*address* or `size` isn't page-aligned.
[K_EROUND]	*lcdesc->startOffset* or *lcdesc->size* isn't fragment-aligned.
[K_EROUND]	*segdesc->startOffset* or *segdesc->size* isn't fragment-aligned.
[K_EADDR]	Some or all the addresses from the target address range are invalid.
[K_EOFFSET]	Tried to flush a segment outside the valid offset range in a segment, as returned by *vmStat*.
[K_EINVAL]	The *flags* argument contains invalid flag values.
[K_EBUSY]	Tried to invalidate or destroy a no–demand (mapped to a region with K_NODEMAND attribute) physical memory.
[K_EFAIL]	An *ipcCall* transaction failed during the remote *lcFlush* or *lcSetRights*.
[K_EMAPPER]	The mapper doesn't respect the vm / mapper protocol.

RESTRICTIONS

The target actor and the current actor must be located on the same site (*vmFlush* only).

The *sgFlush* and *sgSyncAll* calls remain in the interface for backward compatibility. They will be removed in a future release.

ATTRIBUTES

See `attributes`(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

SEE ALSO

`MpGetAccess`(2SEG) , `MpPullIn`(2SEG) , `MpPushOut`(2SEG) , `rgnMap`(2SEG) , `lcOpen`(2SEG) , `lcFillZero`(2SEG) , `lcTrunc`(2SEG)

NAME

sgRead, sgWrite – Read data from a segment into an actor address space; Write data from an actor address space into a segment

SYNOPSIS

#include <mem/chMem.h>
int **sgRead**(KnObjDesc * *srcsegdesc*, KnCap * *dstactorcap*, VmAddr *dstaddress*, VmFlags *flags*);

int **sgWrite**(KnObjDesc * *dstsegdesc*, KnCap * *srcactorcap*, VmAddr *srcaddress*, VmFlags *flags*);

FEATURES

MEM_VM

DESCRIPTION

Caution - This system call is strictly reserved for internal use only. It MUST NOT be used by any application.

The *sgRead* function reads data from a segment into an actor address space. The source data is specified by the *srcsegdesc* argument, which points to a *KnObjDesc* structure described in *rgnInit(2SEG)* . The destination is specified by *dstactorcap* - the destination actor capability, and *dstaddress* - the start address in the destination actor address space.

The *sgWrite* function writes data from an actor address space into a segment. The source data is specified by *srcactorcap* - the source actor capability, and *srcaddress* - the data start address in the source actor address space. The destination is specified by the *dstsegdesc* argument which points to a *KnObjDesc* structure described in *rgnInit(2SEG)* .

If *srcactorcap* and/or *dstactorcap* is K_MYACTOR, the current actor is used. If *srcactorcap* and/or *dstactorcap* is K_SUPERVISOR, the supervisor address space is used.

The *flags* argument is a combination of the following options:

K_MOVE This option indicates that after the operation the contents of the source, from the start to the end of the target data, may be undefined. In this case, the system will try to remap the physical pages containing the data rather than copying them.

K_MOVEAL This option indicates that even if the target data size is not aligned on a page boundary (see *vmPageSize(2K)*), the last page partially covered by the data may be remapped. This means that after the operation the contents of the source, from the start to the next page boundary following the end of the target data, and the contents of the destination, from the end of the target data to the next page boundary, may be undefined.

If any error occurs during an *sgRead* or *sgWrite* operation, the number of bytes read or written before the error is returned in the `size` field of the corresponding *KnObjDesc* structure.

RETURN VALUE

If successful K_OK is returned, otherwise a negative error code is returned.

ERRORS

[K_EADDR]	The address is out of any allocated region.
[K_EFAULT]	Some of the arguments provided are outside the caller's address space.
[K_EINVAL]	An inconsistent actor capability was given.
[K_ESIZE]	The *startOffset* and `size` fields of the *KnObjDesc* structure are inconsistent.
[K_EOFFSET]	Attempted to map a segment outside its valid offset range returned by *vmStat*.
[K_EPROT]	Attempted to write to a read only region.
[K_EUNKNOWN]	*srcactorcap* or *dstactorcap* does not specify a reachable actor.

RESTRICTIONS

The target actor, the source actor and the current actor must be located on the same site.

ATTRIBUTES

See `attributes`(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

SEE ALSO

`rgnInit`(2SEG)

NAME

lcStat, sgStat – get the statistics of a local cache

SYNOPSIS

```
#include <mem/chMem.h>
#include <mem/chMapper.h>
int lcStat(KnCap * lccap, KnLcStat * stat);

int sgStat(KnCap * segcap, KnLcStat * stat);
```

FEATURES

MEM_VM

DESCRIPTION

Caution - This system call is strictly reserved for internal use only. It MUST NOT be used by any application.

The *lcStat* and *sgStat* functions get the statistics of the local cache specified either by *lccap* - a local cache capability, or by *segcap* - a segment capability. In the latter case, the statistics of the cache of the segment on the current site are returned.

The *KnLcStat* structure describes the statistics associated with a local cache, as follows:

```
VmSize   physMem ;
VmSize   lockMem ;
KnCap    segcap ;
KnCap    lccap ;
```

The *physMem* field specifies the physical memory size currently allocated for the local cache.

The *lockMem* field specifies the physical memory size currently fixed for the local cache.

The *segcap* field specifies the capability of the corresponding segment. It is returned by the *lcStat* call only when the caller is a system actor or when the current thread executes in privileged mode.

The *lccap* field specifies the capability of the segment's local cache on the current site. It is returned by the *sgStat* call only when the caller is a system actor or when the current thread executes in privileged mode.

RETURN VALUE

If successful K_OK is returned, otherwise a negative error code is returned.

ERRORS

[K_EFAULT]	Some of the arguments provided are outside the caller's address space.
[K_EUNDEF]	The segment specified is not cached on the site.

ATTRIBUTES

See `attributes`(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

SEE ALSO `lcFlush`(2SEG)

NAME

lcFlush, lcSetRights, sgFlush, sgSyncAll, vmFlush – flush local cache(s)

SYNOPSIS

```
#include <mem/chMem.h>
int lcFlush(KnObjDesc * lcdesc, VmFlags flags, unsigned long ordernb);

int lcSetRights(KnObjDesc * lcdesc, VmFlags flags, unsigned long ordernb);

int sgFlush(KnObjDesc * segdesc, VmFlags flags);

int sgSyncAll(void);

int vmFlush(KnCap * actorcap, VmAddr address, VmSize size, VmFlags flags);
```

FEATURES

MEM_VM

DESCRIPTION

Caution - This system call is strictly reserved for internal use only. It MUST NOT be used by any application.

The *lcFlush* call performs a flush operation on a range of a local cache. The *lcdesc* argument points to a *KnObjDesc* structure whose members are the following:

```
KnCap     dataObject ;
VmOffset  startOffset ;
VmSize    size ;
```

where the *dataObject* field is the local cache capability, the *startOffset* field is the range start offset in the local cache, and the `size` field is the range size. Both *startOffset* and `size` must be fragment-aligned. The size of the fragment is implementation–dependent and is usually equal to the virtual page size divided by 8 (number of bits in one byte).

For *lcFlush*, the target local cache is directly specified by its capability; the flush operation is applied even if the cache is managed by a remote site.

The *sgFlush* call takes the same arguments, except the *dataObject* field specifies the capability of the cached segment. For *sgFlush*, the target local cache is indirectly specified by the capability of the corresponding segment; the flush operation implicitly applies to the segment's cache(s) located on the caller's site.

The *vmFlush* call performs a flush operation on the ranges of the local caches mapped to an address range of an actor address space. The address range must be page-aligned.

The target actor is specified by *actorcap* - a pointer to the actor capability. If *actorcap* is K_MYACTOR, the address space of the current actor is used. If *actorcap* is K_SVACTOR, the supervisor address space is used.

The *flags* argument specifies the mode of the flush and the upper access rights for the target parts after the flush operation.

If the K_COPYBACK flag is set, *xxFlush* writes back any (even clean) cached data of the target parts and keeps the parts access rights unchanged.

If the K_WRITABLE flag is set, *xxFlush* writes back all modified data of the target parts and keeps the parts access rights unchanged.

If the K_READABLE flag is set, *xxFlush* writes back all modified data of the target parts and then sets the parts to read access only.

If the K_FREEZE flag is set, *xxFlush* writes back all modified data of the target parts and then sets the parts as non-accessible.

If the K_NOACCESS flag is set, *xxFlush* writes back all modified data of the target parts and invalidates them.

If the K_DESTROY flag is set, *xxFlush* invalidates the target parts without writing back.

If the K_ASYNC flag is set, the vm performs the write operations required by the *xxFlush* asynchronously.

If K_PAGEFAULT flag is set, *xxFlush* is a result of a page fault. This type of *lcFlush* operation could break the atomicity of a sgRead / sgWrite operation running concurrently on the same local cache parts.

The K_ORDERED flag can only be used with an *lcFlush* or an *lcSetRights* request. If the flags is set, the *orderNb* argument specifies the *lcFlush* message order number. It allows the kernel to detect the situation when two mesages sent by a mapper in one order are received by the kernel in another. For instance, if the mapper grants certain access rights in a *MpGetAccess* reply first, then, processing another request, calls an *lcFlush* to recall the access rights, considering the order numbers, the kernel is aware that the access returned by the *MpGetAccess* reply was already recalled by the mapper, and performs another *MpGetAccess* request.

The *lcSetRights* operation is similar to the *lcFlush* operation, except that it only changes data protections without invalidation and / or writing back: the K_WRITABLE flag is ignored, the K_READABLE flag sets the target parts read—only, whereas the K_FREEZE and K_NOACCESS flags set the target parts to non-accessible. The K_COPYBACK, K_ASYNC and K_DESTROY flags are prohibited.

The *sgSyncAll* operation writes back asynchronously all dirty parts of all local caches on the site, except the local caches mapped at least once to a region with the K_NOSYNC attribute (see *rgnMap(2SEG)).*

RETURN VALUE

If successful K_OK is returned, otherwise a negative error code is returned.

ERRORS

[K_EFAULT]	Some of the arguments provided are outside the caller's address space.

[K_EINVAL]	An inconsistent actor capability was provided.
[K_EUNKNOWN]	*actorcap* does not specify a reachable actor.
[K_EROUND]	*address* or `size` isn't page-aligned.
[K_EROUND]	*lcdesc->startOffset* or *lcdesc->size* isn't fragment-aligned.
[K_EROUND]	*segdesc->startOffset* or *segdesc->size* isn't fragment-aligned.
[K_EADDR]	Some or all the addresses from the target address range are invalid.
[K_EOFFSET]	Tried to flush a segment outside the valid offset range in a segment, as returned by *vmStat*.
[K_EINVAL]	The *flags* argument contains invalid flag values.
[K_EBUSY]	Tried to invalidate or destroy a no–demand (mapped to a region with K_NODEMAND attribute) physical memory.
[K_EFAIL]	An *ipcCall* transaction failed during the remote *lcFlush* or *lcSetRights*.
[K_EMAPPER]	The mapper doesn't respect the vm / mapper protocol.

RESTRICTIONS

The target actor and the current actor must be located on the same site (*vmFlush* only).

The *sgFlush* and *sgSyncAll* calls remain in the interface for backward compatibility. They will be removed in a future release.

ATTRIBUTES

See `attributes`(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

SEE ALSO

`MpGetAccess`(2SEG) , `MpPullIn`(2SEG) , `MpPushOut`(2SEG) , `rgnMap`(2SEG) , `lcOpen`(2SEG) , `lcFillZero`(2SEG) , `lcTrunc`(2SEG)

NAME

sgRead, sgWrite – Read data from a segment into an actor address space; Write data from an actor address space into a segment

SYNOPSIS

#include <mem/chMem.h>
int **sgRead**(KnObjDesc * *srcsegdesc*, KnCap * *dstactorcap*, VmAddr *dstaddress*, VmFlags *flags*);

int **sgWrite**(KnObjDesc * *dstsegdesc*, KnCap * *srcactorcap*, VmAddr *srcaddress*, VmFlags *flags*);

FEATURES

MEM_VM

DESCRIPTION

Caution - This system call is strictly reserved for internal use only. It MUST NOT be used by any application.

The *sgRead* function reads data from a segment into an actor address space. The source data is specified by the *srcsegdesc* argument, which points to a *KnObjDesc* structure described in *rgnInit(2SEG)* . The destination is specified by *dstactorcap* - the destination actor capability, and *dstaddress* - the start address in the destination actor address space.

The *sgWrite* function writes data from an actor address space into a segment. The source data is specified by *srcactorcap* - the source actor capability, and *srcaddress* - the data start address in the source actor address space. The destination is specified by the *dstsegdesc* argument which points to a *KnObjDesc* structure described in *rgnInit(2SEG)* .

If *srcactorcap* and/or *dstactorcap* is K_MYACTOR, the current actor is used. If *srcactorcap* and/or *dstactorcap* is K_SUPERVISOR, the supervisor address space is used.

The *flags* argument is a combination of the following options:

K_MOVE This option indicates that after the operation the contents of the source, from the start to the end of the target data, may be undefined. In this case, the system will try to remap the physical pages containing the data rather than copying them.

K_MOVEAL This option indicates that even if the target data size is not aligned on a page boundary (see *vmPageSize(2K)*), the last page partially covered by the data may be remapped. This means that after the operation the contents of the source, from the start to the next page boundary following the end of the target data, and the contents of the destination, from the end of the target data to the next page boundary, may be undefined.

If any error occurs during an *sgRead* or *sgWrite* operation, the number of bytes read or written before the error is returned in the `size` field of the corresponding *KnObjDesc* structure.

RETURN VALUE

If successful K_OK is returned, otherwise a negative error code is returned.

ERRORS

[K_EADDR]	The address is out of any allocated region.
[K_EFAULT]	Some of the arguments provided are outside the caller's address space.
[K_EINVAL]	An inconsistent actor capability was given.
[K_ESIZE]	The *startOffset* and `size` fields of the *KnObjDesc* structure are inconsistent.
[K_EOFFSET]	Attempted to map a segment outside its valid offset range returned by *vmStat*.
[K_EPROT]	Attempted to write to a read only region.
[K_EUNKNOWN]	*srcactorcap* or *dstactorcap* does not specify a reachable actor.

RESTRICTIONS

The target actor, the source actor and the current actor must be located on the same site.

ATTRIBUTES

See `attributes`(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

SEE ALSO

`rgnInit`(2SEG)

NAME

lcFlush, lcSetRights, sgFlush, sgSyncAll, vmFlush – flush local cache(s)

SYNOPSIS

```
#include <mem/chMem.h>
int lcFlush(KnObjDesc * lcdesc, VmFlags flags, unsigned long ordernb);

int lcSetRights(KnObjDesc * lcdesc, VmFlags flags, unsigned long ordernb);

int sgFlush(KnObjDesc * segdesc, VmFlags flags);

int sgSyncAll(void);

int vmFlush(KnCap * actorcap, VmAddr address, VmSize size, VmFlags flags);
```

FEATURES

MEM_VM

DESCRIPTION

Caution - This system call is strictly reserved for internal use only. It MUST NOT be used by any application.

The *lcFlush* call performs a flush operation on a range of a local cache. The *lcdesc* argument points to a *KnObjDesc* structure whose members are the following:

```
KnCap     dataObject ;
VmOffset  startOffset ;
VmSize    size ;
```

where the *dataObject* field is the local cache capability, the *startOffset* field is the range start offset in the local cache, and the `size` field is the range size. Both *startOffset* and `size` must be fragment-aligned. The size of the fragment is implementation–dependent and is usually equal to the virtual page size divided by 8 (number of bits in one byte).

For *lcFlush*, the target local cache is directly specified by its capability; the flush operation is applied even if the cache is managed by a remote site.

The *sgFlush* call takes the same arguments, except the *dataObject* field specifies the capability of the cached segment. For *sgFlush*, the target local cache is indirectly specified by the capability of the corresponding segment; the flush operation implicitly applies to the segment's cache(s) located on the caller's site.

The *vmFlush* call performs a flush operation on the ranges of the local caches mapped to an address range of an actor address space. The address range must be page-aligned.

The target actor is specified by *actorcap* - a pointer to the actor capability. If *actorcap* is K_MYACTOR, the address space of the current actor is used. If *actorcap* is K_SVACTOR, the supervisor address space is used.

The *flags* argument specifies the mode of the flush and the upper access rights for the target parts after the flush operation.

If the K_COPYBACK flag is set, *xxFlush* writes back any (even clean) cached data of the target parts and keeps the parts access rights unchanged.

If the K_WRITABLE flag is set, *xxFlush* writes back all modified data of the target parts and keeps the parts access rights unchanged.

If the K_READABLE flag is set, *xxFlush* writes back all modified data of the target parts and then sets the parts to read access only.

If the K_FREEZE flag is set, *xxFlush* writes back all modified data of the target parts and then sets the parts as non-accessible.

If the K_NOACCESS flag is set, *xxFlush* writes back all modified data of the target parts and invalidates them.

If the K_DESTROY flag is set, *xxFlush* invalidates the target parts without writing back.

If the K_ASYNC flag is set, the vm performs the write operations required by the *xxFlush* asynchronously.

If K_PAGEFAULT flag is set, *xxFlush* is a result of a page fault. This type of *lcFlush* operation could break the atomicity of a sgRead / sgWrite operation running concurrently on the same local cache parts.

The K_ORDERED flag can only be used with an *lcFlush* or an *lcSetRights* request. If the flags is set, the *orderNb* argument specifies the *lcFlush* message order number. It allows the kernel to detect the situation when two mesages sent by a mapper in one order are received by the kernel in another. For instance, if the mapper grants certain access rights in a *MpGetAccess* reply first, then, processing another request, calls an *lcFlush* to recall the access rights, considering the order numbers, the kernel is aware that the access returned by the *MpGetAccess* reply was already recalled by the mapper, and performs another *MpGetAccess* request.

The *lcSetRights* operation is similar to the *lcFlush* operation, except that it only changes data protections without invalidation and / or writing back: the K_WRITABLE flag is ignored, the K_READABLE flag sets the target parts read—only, whereas the K_FREEZE and K_NOACCESS flags set the target parts to non-accessible. The K_COPYBACK, K_ASYNC and K_DESTROY flags are prohibited.

The *sgSyncAll* operation writes back asynchronously all dirty parts of all local caches on the site, except the local caches mapped at least once to a region with the K_NOSYNC attribute (see *rgnMap(2SEG)*).

RETURN VALUE

If successful K_OK is returned, otherwise a negative error code is returned.

ERRORS

[K_EFAULT]	Some of the arguments provided are outside the caller’s address space.

[K_EINVAL]	An inconsistent actor capability was provided.
[K_EUNKNOWN]	*actorcap* does not specify a reachable actor.
[K_EROUND]	*address* or `size` isn't page-aligned.
[K_EROUND]	*lcdesc->startOffset* or *lcdesc->size* isn't fragment-aligned.
[K_EROUND]	*segdesc->startOffset* or *segdesc->size* isn't fragment-aligned.
[K_EADDR]	Some or all the addresses from the target address range are invalid.
[K_EOFFSET]	Tried to flush a segment outside the valid offset range in a segment, as returned by *vmStat*.
[K_EINVAL]	The *flags* argument contains invalid flag values.
[K_EBUSY]	Tried to invalidate or destroy a no–demand (mapped to a region with K_NODEMAND attribute) physical memory.
[K_EFAIL]	An *ipcCall* transaction failed during the remote *lcFlush* or *lcSetRights*.
[K_EMAPPER]	The mapper doesn't respect the vm / mapper protocol.

RESTRICTIONS

The target actor and the current actor must be located on the same site (*vmFlush* only).

The *sgFlush* and *sgSyncAll* calls remain in the interface for backward compatibility. They will be removed in a future release.

ATTRIBUTES

See `attributes`(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

SEE ALSO

`MpGetAccess`(2SEG) , `MpPullIn`(2SEG) , `MpPushOut`(2SEG) , `rgnMap`(2SEG) , `lcOpen`(2SEG) , `lcFillZero`(2SEG) , `lcTrunc`(2SEG)

Index

M

P

R

S

V